

WebSphere Application Server V6 Scalability and Performance Handbook

WebSphere Handbook Series

**Workload manage Web server,
servlet, and EJB requests**

**Learn about performance
monitoring and tuning**



Birgit Roehm
Gang Chen
Andre de Oliveira Fernandes
Cristiane Ferreira
Rodney Krick
Denis Ley
Robert R. Peterson
Gerhard Poul
Joshua Wong
Ruibin Zhou



International Technical Support Organization

WebSphere Application Server V6 Scalability and Performance Handbook

May 2005

Archived

Note: Before using this information and the product it supports, read the information in “Notices” on page xvii.

First Edition (May 2005)

This edition applies to IBM WebSphere Application Server Network Deployment V6 for distributed platforms.

© Copyright International Business Machines Corporation 2005. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	xvii
Trademarks	xviii
Preface	xix
The team that wrote this redbook	xx
Become a published author	xxiv
Comments welcome	xxv
Summary of changes	xxvii
May 2005, First Edition	xxvii
Part 1. Getting started	1
Chapter 1. Overview and key concepts	3
1.1 Objectives	4
1.1.1 Scalability	5
1.1.2 Workload management	6
1.1.3 Availability	7
1.1.4 Maintainability	9
1.1.5 Session state	10
1.1.6 Performance impact of WebSphere Application Server security ...	10
1.2 WebSphere Application Server architecture	12
1.2.1 WebSphere Application Server Network Deployment components .	12
1.2.2 Web clients	15
1.2.3 Java clients	16
1.3 Workload management	16
1.3.1 Web server workload management	17
1.3.2 Plug-in workload management	17
1.3.3 Workload management using WebSphere clustering	19
1.3.4 Enterprise Java Services workload management	23
1.4 Managing session state among servers	24
1.4.1 HTTP sessions and the session management facility	25
1.4.2 EJB sessions or transactions	28
1.4.3 Server affinity	30
1.5 Performance improvements over previous versions	31
1.6 The structure of this redbook	35
Chapter 2. Infrastructure planning and design	39
2.1 Infrastructure deployment planning	40

2.1.1 IBM Design Centers for e-business on demand	41
2.2 Design for scalability	42
2.2.1 Understanding the application environment	42
2.2.2 Categorizing your workload.	44
2.2.3 Determining the most affected components	50
2.2.4 Selecting the scaling techniques to apply	51
2.2.5 Applying the technique(s)	56
2.2.6 Re-evaluating	57
2.3 Sizing.	58
2.4 Benchmarking	59
2.4.1 IBM eServer™ Benchmarking Centers.	60
2.4.2 IBM Test Center	61
2.5 Performance tuning.	62
2.5.1 Application design problems	62
2.5.2 Understand your requirements	63
2.5.3 Test environment setup.	63
2.5.4 Test phases.	64
2.5.5 Load factors	66
2.5.6 Production system tuning	66
2.5.7 Conclusions.	68
Chapter 3. Introduction to topologies	71
3.1 J2EE tiers model	72
3.2 Topology selection criteria.	74
3.3 Strategies for scalability	74
3.4 Web server topology in a Network Deployment cell	78
3.4.1 Web server managed node.	79
3.4.2 Web server unmanaged node.	79
3.4.3 IBM HTTP Server (IHS) as unmanaged node (special case)	80
3.5 Single machine, single node, Web server separated	81
3.6 Vertical scaling topology	82
3.7 Horizontal scaling topology	82
3.8 Horizontal scaling with IP sprayer topology.	84
3.9 Topology with redundancy of several components	85
3.10 The sample topology.	86
3.11 Topologies and high availability	89
3.11.1 Using WebSphere Load Balancer custom advisor	91
3.12 Topology selection summary.	94
Part 2. Distributing the workload	97
Chapter 4. Introduction to WebSphere Edge Components	99
4.1 Introduction	100
4.1.1 Scalability	100

4.1.2 Availability	100
4.1.3 Performance	100
4.2 IBM WebSphere Edge Components overview	101
4.3 Load Balancer overview	102
4.3.1 Dispatcher	102
4.3.2 Content Based Routing (CBR)	109
4.3.3 Site Selector	110
4.3.4 Cisco CSS Controller and Nortel Alteon Controller	110
4.4 Server affinity in Load Balancer	110
4.4.1 Stickyness to source IP address	111
4.4.2 Cross port affinity	112
4.4.3 Passive cookie affinity	113
4.4.4 Active cookie affinity	113
4.4.5 URI affinity	113
4.4.6 SSL session ID	114
4.5 Load Balancer topologies	114
4.5.1 Load Balancer on a dedicated server	114
4.5.2 Collocated servers	115
4.5.3 High availability	116
4.5.4 Mutual high availability	118
4.6 Dispatcher scripts	119
4.7 Load Balancer features comparison	120
4.8 Caching Proxy overview	122
4.8.1 Forward proxy	122
4.8.2 Reverse proxy (IP forwarding)	123
4.8.3 Using multiple Caching Proxy servers	124
4.8.4 Dynamic caching	124
4.9 WebSphere Edge Components V6 new features	125
Chapter 5. Using IBM WebSphere Edge Components	127
5.1 Load Balancer installation	128
5.1.1 Load Balancer installation wizard	128
5.1.2 Load Balancer installation using SMIT in AIX	132
5.1.3 Post installation tasks	134
5.2 Load Balancer configuration: basic scenario	135
5.2.1 Configuring the Load Balancer cluster	136
5.2.2 Configuring the balanced servers	148
5.2.3 Testing the basic scenario	156
5.3 Load Balancer: high availability scenario	162
5.3.1 Configuring high availability	162
5.3.2 Adding reach targets	169
5.3.3 Checking the configuration	170
5.3.4 Configuring the high availability scripts	171

5.3.5 Testing the high availability scenario	177
5.4 Load Balancer: NAT scenario	181
5.4.1 Testing the NAT scenario	189
5.5 Load Balancer: additional configuration options	190
5.5.1 Basic Load Balancer scenario with customizable advisor settings	190
5.5.2 Using WebSphere Application Server custom advisor	193
5.5.3 Starting Dispatcher automatically after a reboot	203
5.5.4 Starting and stopping Dispatcher components	204
5.6 Caching Proxy installation	204
5.6.1 Checking prerequisites	205
5.6.2 Caching Proxy installation wizard	206
5.6.3 Caching Proxy installation using SMIT in AIX	210
5.7 Caching Proxy configuration	212
5.7.1 Using the Caching Proxy configuration wizard	213
5.7.2 Using the Caching Proxy Web-based administration tool	216
5.7.3 Manual configuration	218
5.7.4 Creating and defining a cache storage	219
5.8 Managing the Caching Proxy process	222
5.8.1 Testing the Caching Proxy scenario	224
Chapter 6. Plug-in workload management and failover	227
6.1 Introduction	228
6.2 WebContainer transport chains and virtual hosts	230
6.2.1 WebContainer Inbound Chains	231
6.2.2 Virtual hosts	232
6.2.3 Transport chains: the details	234
6.3 Creating clusters and cluster members	244
6.4 Web server topologies	244
6.4.1 Managed Web servers	246
6.4.2 Unmanaged Web servers	248
6.4.3 Unmanaged IBM HTTP Server V6.0 server (special case)	250
6.5 WebSphere plug-in configuration file	251
6.5.1 The plug-in configuration file	252
6.5.2 Generation of the plug-in configuration file	258
6.5.3 Propagation of the plug-in file	262
6.5.4 Bypassing the plug-in	263
6.6 WebSphere plug-in workload management	264
6.6.1 Processing requests	264
6.6.2 Plug-in workload management policies	267
6.7 Web container failures and failover	272
6.7.1 Primary and backup servers	275
6.8 HTTP session management	279

6.8.1 Session affinity	281
6.8.2 Session identifiers	282
6.8.3 Session management and failover inside the plug-in	285
6.8.4 Session management configuration	287
6.8.5 Database session management configuration	291
6.8.6 Memory-to-memory replication configuration	294
6.8.7 Understanding DRS (Data Replication Services)	297
6.8.8 Session management tuning	300
6.9 Troubleshooting the Web server plug-in	304
6.9.1 Logging	304
6.9.2 Trace	308
6.10 WebSphere plug-in behavior	309
6.10.1 Normal operation	310
6.10.2 Failover operation	320
6.10.3 Tuning failover	335
Chapter 7. EJB workload management	341
7.1 Enabling EJB workload management	342
7.2 EJB types and workload management	343
7.2.1 Stateless session beans	344
7.2.2 Stateful session beans	344
7.2.3 Entity beans	345
7.3 EJB bootstrapping	347
7.3.1 Bootstrapping within WebSphere containers	347
7.3.2 Bootstrapping outside of a J2EE container	352
7.4 How EJBs participate in workload management	354
7.4.1 Initial request	355
7.4.2 Subsequent requests	356
7.4.3 Cluster run state changes	357
7.5 EJB workload management routing policy	358
7.5.1 Server weighted round robin	359
7.5.2 Prefer local	366
7.5.3 Process affinity	370
7.5.4 Transaction affinity	370
7.6 EJB high availability and failover	371
7.6.1 EJB client redundancy and bootstrap failover support	371
7.6.2 EJB container redundancy and EJB WLM failover support	372
7.6.3 EJB failover behavior	374
Part 3. Implementing the solution	385
Chapter 8. Implementing the sample topology	387
8.1 Overview	388
8.2 Software products	388

8.2.1	The sample topology	388
8.2.2	Applications used in our sample topology	391
8.3	Installation summary	392
8.4	Installing and configuring WebSphere Edge Components	393
8.4.1	Configuring the Caching Proxy	394
8.4.2	Configuring the Load Balancer	394
8.4.3	Checking the Load Balancer and Caching Proxy configurations	395
8.5	Installing WebSphere and configuring clusters	395
8.5.1	Introduction	395
8.5.2	Deployment Manager installation and profile creation	396
8.5.3	Application server nodes installation (federated nodes)	400
8.5.4	Verifying the profiles	402
8.5.5	Creating the Web container cluster	404
8.5.6	Creating the EJB cluster	407
8.5.7	Verifying the cluster topology	410
8.5.8	Configure distributed session management	411
8.5.9	Starting the clusters	415
8.6	Installing and configuring IBM HTTP Server 6.0	416
8.6.1	IBM HTTP Server 6.0 installation	416
8.6.2	WebSphere plug-in installation	417
8.6.3	Configuring Web servers in the cell	420
8.6.4	Testing Web server configurations	426
8.7	Installing and configuring BeenThere	426
8.7.1	BeenThere installation summary	427
8.7.2	Install BeenThere	427
8.7.3	Regenerate Web server plug-in	431
8.7.4	Configuring WEBcluster members for BeenThere	432
8.7.5	Verifying BeenThere	434
8.8	Installing and configuring Trade 6	436
8.8.1	Download the Trade 6.0.1 installation package	438
8.8.2	Set up and configure tradedb database	438
8.8.3	Configure the WebSphere cell	440
8.8.4	Install Trade 6 from the WebSphere Administrative Console	450
8.8.5	Regenerate Web server plug-in and start servers	453
8.8.6	Install Trade 6 using the installation script	453
8.8.7	Working with Trade 6	457
8.8.8	Verify failover with Trade 6	460
8.8.9	Volume testing Trade 6	460
8.8.10	Uninstalling Trade 6	460
Part 4.	High availability and caching	463
Chapter 9.	WebSphere HAManager	465

9.1	Introduction	466
9.2	Core group	467
9.2.1	Core group coordinator	468
9.2.2	Transport buffer	472
9.2.3	Distribution and Consistency Services	474
9.2.4	Core group policy	474
9.2.5	Match criteria	477
9.2.6	Transport type	478
9.3	High availability group	479
9.4	Discovery of core group members	482
9.5	Failure Detection	483
9.5.1	Active failure detection	483
9.5.2	TCP KEEP_ALIVE	484
9.6	JMS high availability	485
9.7	Transaction Manager high availability	485
9.7.1	Transaction Manager HA of previous versions of WebSphere	487
9.7.2	Hot-failover of Transaction Manager using shared file system	489
9.7.3	Hot-failover of transaction logs using external HA software	496
Chapter 10.	Dynamic caching	501
10.1	Introduction	502
10.1.1	WWW caching services	502
10.1.2	Fragment caching	506
10.1.3	Dynamic caching scenarios	507
10.2	What is new in WebSphere V6 dynamic caching	509
10.2.1	Dynamic Content Provider interface	509
10.2.2	Cache instances	509
10.2.3	Caching Struts and Tiles applications	510
10.2.4	Cache replication	511
10.3	The cachespec.xml configuration file	511
10.3.1	cachespec.xml elements	512
10.3.2	Dynamic Cache Policy Editor	515
10.4	Using WebSphere dynamic cache service	515
10.4.1	Installing Dynamic Cache Monitor	516
10.4.2	Enabling dynamic cache service	524
10.5	WebSphere dynamic caching scenarios	528
10.5.1	Servlet/JSP result caching	529
10.5.2	Struts and Tiles caching	537
10.5.3	Command caching	540
10.5.4	Cache replication	547
10.5.5	Cache invalidation	556
10.5.6	Troubleshooting the dynamic cache service	557
10.6	WebSphere external caching scenarios	558

10.6.1	WebSphere External Cache configuration	560
10.6.2	External caching by Web server plug-in	566
10.6.3	External caching on the IBM HTTP Server	574
10.6.4	External caching on the Caching Proxy	578
10.7	Using the Dynamic Cache Policy Editor	586
10.7.1	Dynamic Cache Policy Editor installation	586
10.7.2	Creating cache policy entries	590
10.7.3	Examples: Creating cachespec.xml entries with the Dynamic Cache Policy Editor	592
10.8	Conclusion	609
10.9	Benchmarking Trade 3	611
10.9.1	Dynamic caching	611
10.9.2	Edge Side Includes	614
10.10	Reference	616
Part 5	Messaging	619
Chapter 11	Using asynchronous messaging for scalability and performance	621
11.1	Introduction	622
11.2	Basic use of the JMS API	622
11.2.1	The unified programming interface	622
11.2.2	Consuming JMS messages	625
11.3	Choosing what format to use within JMS messages	625
11.4	Managing workload for asynchronous messaging	627
11.4.1	Basic workload patterns	627
11.4.2	Selectors	637
11.4.3	Application defined persistence	641
11.4.4	Freeing JMS object resources	641
Chapter 12	Using and optimizing the default messaging provider	643
12.1	Introduction	644
12.2	Introduction to the Service Integration Bus and the default messaging provider	645
12.2.1	Bus or Service Integration Bus	645
12.2.2	Bus members and messaging engines	646
12.2.3	Destinations	647
12.2.4	JMS activation specification	647
12.2.5	Message reliability	648
12.2.6	Data stores	649
12.3	Components used in a default messaging provider configuration	649
12.3.1	JMS component diagram for sending a message	650
12.3.2	Bus and message-driven beans	652
12.4	Component relationships	655

12.5	Clustering, high availability and workload management	656
12.5.1	Cluster bus members for high availability	656
12.5.2	Cluster bus members for workload management	657
12.5.3	Partitioned queues	657
12.5.4	JMS clients connecting into a cluster of messaging engines	659
12.5.5	Preferred servers and core group policies	660
12.6	Choosing optimal configuration settings	663
12.6.1	Important connection factory settings	663
12.6.2	Setting up default messaging provider connection pools	667
12.6.3	Important JMS activation specification settings	670
12.7	Failure to process a message	671
12.8	Usage scenarios: Trade 6 and the default messaging provider	672
12.8.1	What does Trade 6 use the default messaging provider for?	673
12.8.2	Example 1: One messaging engine on the bus	674
12.8.3	Example 2: One messaging engine per server	686
12.9	Workload management example using BeenThere	691
12.9.1	Taking advantage of the BeenThere documentation	691
12.9.2	Possible topologies to use with BeenThere	692
12.10	Monitoring performance with Tivoli Performance Viewer	693
Chapter 13.	Understanding and optimizing the use of WebSphere MQ	697
13.1	Introduction	698
13.1.1	JMS component diagram for sending a message	699
13.1.2	JMS and message-driven beans	702
13.2	MQ JMS component relationships	705
13.2.1	Component relationships when using MDBs	705
13.3	Choosing optimal configuration settings	708
13.3.1	Creation of the MQ provider objects at the correct scope	708
13.3.2	Important MQ JMS component settings	711
13.3.3	The listener service and listener ports	718
13.3.4	Setting up the connection factory pools	724
13.3.5	More information	725
13.4	JMS Listener port failure behavior	726
13.4.1	Failure in the listener port	726
13.4.2	Failure to process a message	728
13.5	Example JMS topologies and scenarios	729
13.5.1	What does Trade 6 use JMS for?	730
13.5.2	Clustered Trade 6 with WebSphere MQ and WebSphere Business Integration Event Broker	732
13.6	Monitoring performance with Tivoli Performance Viewer	765
13.6.1	What do the counters under JCA Connection Pools mean?	766

Part 6.	Performance monitoring, tuning, and coding practices	767
----------------	---	------------

Chapter 14. Server-side performance and analysis tools	769
14.1 The dimensions of monitoring	770
14.1.1 Overview: Collecting and displaying application server data	771
14.2 Performance Monitoring Infrastructure	771
14.2.1 Performance data classification	773
14.2.2 Performance data hierarchy	774
14.2.3 Performance data counters	779
14.2.4 PMI predefined statistic sets	780
14.2.5 Enabling the PMI service	782
14.2.6 Using JVMPi facility for PMI statistics	786
14.2.7 Summary	790
14.3 Using Tivoli Performance Viewer	790
14.3.1 About Tivoli Performance Viewer	791
14.3.2 What can Tivoli Performance Viewer do?	792
14.3.3 Starting Tivoli Performance Viewer	793
14.3.4 Configuring Tivoli Performance Viewer	794
14.3.5 Tivoli Performance Viewer summary reports	796
14.3.6 Displaying by performance modules	798
14.3.7 Getting online help	803
14.4 Other performance monitoring and management solutions	803
14.5 Developing your own monitoring application	804
14.6 Request Metrics	805
14.6.1 Enabling and configuring Request Metrics	806
14.6.2 Request Metrics trace record format	808
14.6.3 Filters	812
14.7 Performance Advisors	813
14.7.1 Runtime Performance Advisor configuration settings	815
14.7.2 Advice configuration settings	816
14.7.3 Using the Runtime Performance Advisor	817
14.7.4 Runtime Performance Advisor output	819
14.7.5 Using TPV Advisor	821
14.7.6 TPV Advisor output	824
14.8 Dynamic Cache Monitor	826
14.9 Monitoring the IBM HTTP Server	826
14.10 Log Analyzer	828
14.11 Application management	829
14.11.1 Tivoli Monitoring for Transaction Performance V5.3 (TMTP)	831
14.11.2 WebSphere Studio Application Monitor V3.1 (WSAM)	833
14.12 Reference	838
Chapter 15. Development-side performance and analysis tools	839
15.1 Introduction	840
15.2 The Profiler (profiling tools)	840

15.2.1	What's new with profiling	841
15.2.2	Profiling architecture	844
15.2.3	IBM Rational Agent Controller	846
15.2.4	Setting up for Profiling	848
15.2.5	Profiling an application	856
15.2.6	Profiler views	857
15.2.7	Resolving performance bottlenecks - Execution time analysis . . .	873
15.3	IBM Page Detailer	886
15.3.1	Overview	886
15.3.2	Important considerations	890
15.3.3	Key factors	890
15.3.4	Tips for using Page Detailer	891
15.3.5	Reference	894
Chapter 16. Application development: best practices for application design, performance and scalability		895
16.1	Introduction	896
16.2	Presentation layer	898
16.2.1	JavaServer Pages	899
16.2.2	Struts	901
16.2.3	JavaServer Faces	903
16.2.4	XML/XSLT processing	905
16.2.5	Caching for the presentation layer	907
16.3	Control	907
16.3.1	Maintaining state: stateful session beans versus HTTP session .	909
16.4	Business logic layer	912
16.4.1	How to implement the interface to the business logic	913
16.4.2	Facade	915
16.4.3	EJB Command Framework	917
16.4.4	Caching business logic	919
16.5	Data access layer	919
16.5.1	Service Data Objects	920
16.5.2	Entity beans	924
16.5.3	Java Data Objects	928
16.5.4	EJB session bean: direct access to back end	929
16.6	Key factors for application performance	930
16.6.1	Memory	930
16.6.2	Synchronization	933
16.6.3	Logging	934
16.6.4	Database access	935
16.7	General coding issues	936
16.8	Reference	938

Chapter 17. Performance tuning	939
17.1 Testing the performance of an application	940
17.1.1 Introduction to application performance testing	940
17.2 Selecting testing tools	942
17.3 Tools of the trade	945
17.3.1 ApacheBench	945
17.3.2 OpenSTA	948
17.3.3 Rational Performance Tester	956
17.3.4 Other testing tools	964
17.4 Performance monitoring guidelines	965
17.4.1 Top ten monitoring hotlist	965
17.4.2 Performance analysis	969
17.5 Performance tuning guidelines	975
17.5.1 Tuning parameter hotlist	975
17.5.2 Parameters to avoid failures	976
17.5.3 Hardware and capacity settings	976
17.5.4 Adjusting WebSphere Application Server system queues	977
17.5.5 Application assembly performance checklist	991
17.5.6 Java tuning	1002
17.5.7 Operating system tuning	1012
17.5.8 The Web server	1015
17.5.9 Dynamic Cache Service	1020
17.5.10 Security settings	1020
17.5.11 Tuning Secure Sockets Layer	1021
17.5.12 Object Request Broker (ORB)	1022
17.5.13 XML parser selection	1025
17.5.14 DB2 tuning	1026
17.5.15 Additional reference materials	1029
Part 7. Appendices	1031
Appendix A. Sample URL rewrite servlet	1033
Setting up the servlet	1034
Source code	1034
Steps to install SessionSampleURLRewrite servlet	1035
Installing the urltest Web module	1035
Appendix B. Additional material	1037
Locating the Web material	1037
Using the Web material	1038
System requirements for downloading the Web material	1038
How to use the Web material	1038
Related publications	1039

IBM Redbooks 1039

Other publications 1040

Online resources 1040

How to get IBM Redbooks 1047

Help from IBM 1047

Index 1049

Archived

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.


This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

@server®	AFP™	OS/400®
@server®	AIX®	Purify®
Redbooks (logo)  ™	CICS®	Rational Suite®
alphaWorks®	Domino®	Rational®
developerWorks®	DB2®	Redbooks™
e-business on demand™	Extreme Blue™	RS/6000®
eServer™	HACMP™	S/390®
ibm.com®	IBM®	SecureWay®
iSeries™	IMS™	TestStudio®
pSeries®	Lotus®	Tivoli®
xSeries®	Notes®	TotalStorage®
z/OS®	OS/2®	WebSphere®
zSeries®	OS/390®	Workplace™

The following terms are trademarks of other companies:

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Intel Inside (logos), MMX, and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

Preface

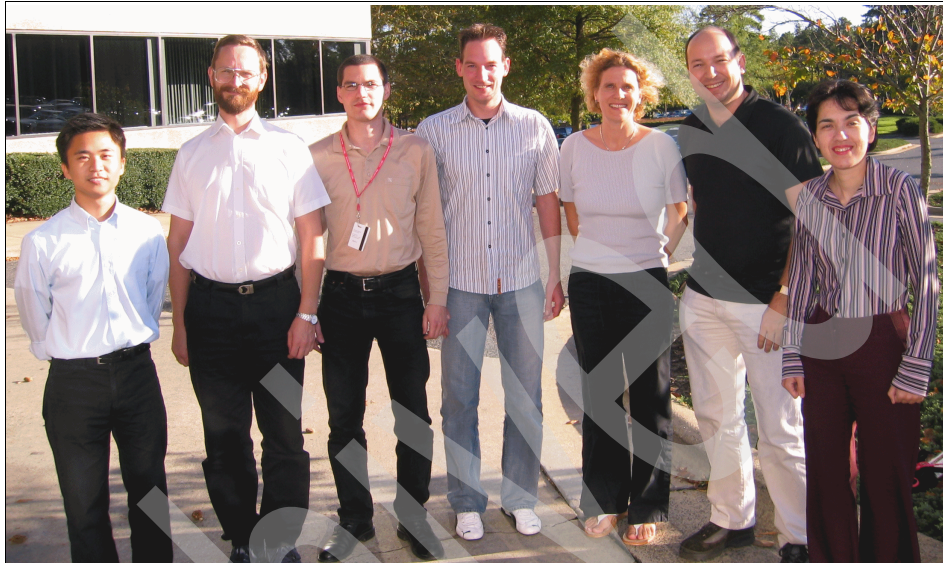
This IBM® Redbook discusses various options for scaling applications based on IBM WebSphere Application Server Network Deployment V6. It explores how a basic WebSphere® configuration can be extended to provide more computing power by better exploiting the power of each machine and by using multiple machines. It examines a number of techniques:

- ▶ Using the IBM WebSphere Edge Components' Load Balancer to distribute the load among multiple Web servers.
- ▶ Using the WebSphere Web server plug-in to distribute the load from one Web server to multiple application servers in a server cluster.
- ▶ Using the WebSphere EJB workload management facility to distribute the load at the EJB level.
- ▶ Using dynamic caching techniques and the IBM WebSphere Edge Components' Caching Proxy to improve the performance of a Web site.
- ▶ Using application development best practices to develop a scalable application.
- ▶ Using the performance tuning options available with WebSphere to adjust the application server configuration to the needs of your application.

This redbook provides step-by-step instructions for implementing a sample, multiple-machine environment. We use this environment to illustrate most of the IBM WebSphere Application Server Network Deployment V6 workload management and scalability features.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.



From left to right: Joshua Wong, Balazs Csepregi-Horvath, Gerhard Poul, Denis Ley, Birgit Roehm, Andre de Oliveira Fernandes, Cristiane Ferreira

Birgit Roehm is a Project Leader at the International Technical Support Organization, Raleigh Center. She writes and teaches workshops about various aspects of WebSphere and Domino®. Before joining the ITSO in 2003, Birgit worked in IBM @server® iSeries™ Advanced Technical Support, Germany, and was responsible for Domino and WebSphere on iSeries.



Gang Chen is a Consulting I/T specialist in IBM Software Services for WebSphere, servicing the New York Metro area. Gang is an expert in helping enterprise customers with their complex transactional requirements. He works with several major Wall Street customers in building mission-critical transactional systems.

Andre de Oliveira Fernandes is a WebSphere specialist working for Banco Central do Brasil (Brazilian Central Bank). After five years of developing J2EE applications, Andre now coordinates the team responsible for Banco Central's WebSphere production environment. Andre also helps to define and enforce development practices and processes that result in a more predictable behavior for critical applications.

Cristiane Ferreira is an IT Specialist working for IBM Global Services in Brazil. She has ten years of experience in UNIX® platforms and TCP/IP networking, and has been working with the WebSphere Support and Services team for the past three years. Her areas of expertise include WebSphere Application Server, WebSphere Edge Components, AIX® and Linux® systems, TCP/IP networking, firewalls and networking security. She is an IBM Certified Advanced System Administrator - WebSphere Application Server V5.0 and IBM Certified Systems Expert - Administration for IBM WebSphere Application Server Advanced Edition V4.0. She is also an AIX Certified Specialist, Linux Professional Institute Certified Level 1 Specialist and Tivoli® Certified Solutions Expert - IBM SecureWay® Firewall for AIX.



Rodney Krick works as a Senior Consultant for aformatik Training & Consulting GmbH & Co KG in Germany. He has 15 years of experience in system design and implementation, coaching and teaching, first in Brazil and, since 1996, in Germany. He teaches Java™, WebSphere Administration and DB2® UDB. He is a Certified Instructor for IBM DB2 Software in ECIS (Education Center for IBM Software), IBM Certified Database Administrator (DB2 UDB V8.1 for Linux, UNIX and Windows®), IBM Certified Solutions Expert (DB2 UDB V7.1 Database Administration for Linux, UNIX, Windows and OS/2®), IBM Certified Solutions Expert (DB2 V7.1 Family Application Development) and IBM Certified Solutions Expert (DB2 UDB V7.1 Database Administration for OS/390®).

Denis Ley is a WebSphere Technical Consultant working for the IBM Innovation Center for Business Partners in Stuttgart, Germany. He has five years of experience with WebSphere Application Server and the corresponding development tools. He holds a degree in Computer Science from the University of Stuttgart. His main area of expertise is consulting on the WebSphere foundation and tools products, in particular best practices, design patterns, building portable applications, J2EE compliance and performance tuning. In the IBM Innovation Center, he assists Independent Software Vendors (ISVs) in developing new WebSphere based solutions or porting their existing applications from different application servers to IBM WebSphere. He also has extensive experience with the architecture and the implementation of e-business/on demand solutions, with a strong focus on performance aspects.



Robert Peterson is a member of IBM Software Services for WebSphere, consultants who help customers achieve success with WebSphere products. He is also an alumnus of the IBM Extreme Blue™ program, during which he built a prototype for a demand-based pricing system for the energy industry. He was chosen to present his work to the CEO and Chairman of the IBM board, Sam Palmisano. He holds a M.S. in Computer Engineering from the University of Florida.

Gerhard Poul is an IT Specialist in IBM Global Services based in Vienna, Austria. He is a specialist for WebSphere Application Server on distributed platforms with knowledge in advanced scalability and performance topics. Gerhard is an IBM Certified Advanced System Administrator (WebSphere Application Server V5.0). His areas of expertise include WebSphere Application Server Security, WebSphere Commerce, WebSphere Portal, and Lotus® Workplace™.

Joshua Wong is an IT Specialist working for IBM Integrated Technology Services, Hong Kong. He has three years of project experience in AIX and WebSphere technologies. Joshua was a member of the IBM team which delivered a high-performing, resilient infrastructure to one of the largest Hong Kong law enforcement agencies. His areas of expertise include AIX HACMP™, WebSphere Application Server, WebSphere Edge Components and JVM tuning. He holds a degree in Computer Science and Information Systems from the University of Hong Kong.



Ruibin Zhou is a staff software engineer in WebSphere Application Server SVT located at RTP, NC. He has four years of experience in developing customer-like J2EE applications to test WebSphere Application Server. His areas of expertise include J2EE, WebSphere Application Server, WebSphere MQ and DB2. He has written extensively on server-side performance tools and the default messaging provider.

A special thank you goes to Balazs Csepregi-Horvath who worked on updating the workshop material of the ITSO course “WebSphere V5.1 Performance and Scalability” for WebSphere V6 while the rest of the team was updating the redbook.

Balazs Csepregi-Horvath is a Customer Engineer working for IBM Global Services in Hungary. He holds a bachelor's degree in Heavy Current Automation and a bachelor's degree in Information Technology, both from Kando Kalman Technical College, Budapest, Hungary. He has five years of expertise supporting WebSphere and DB2 products and is an IBM Certified Advanced Technical Expert for IBM pSeries® and AIX 5L.

A very special thank you goes to David Currie, Martin Phillips, and Martin Smithson, IBM United Kingdom, for their support with the messaging chapters.

Thanks to the following people for their contributions to this project:

Hernan Cunico, John Ganci, Peter Kovari, Linda Robinson, Carla Sadtler, Margaret Ticknor, Jeanne Tucker, Cecilia Bardy
International Technical Support Organization, Raleigh Center

Aleksandr Nartovich
International Technical Support Organization, Rochester Center

Morten Moeller
International Technical Support Organization, Austin Center

Joe DeCarlo
International Technical Support Organization, Almaden Center

Masahiro Ozaki
Mitsubishi Electronics

David Adcox, Christopher Blythe, Andy Chow, Priyanka Jain, Nirmala Kodali, Joel Meyer, Jakob Mickley, Robbie J. Minshall, Michael Morton, Wenjian Qiao, Jeff Robbins, Andrew Spyker, Arvind Srinivasan, Rengan Sundararaman, Sharon Weed
IBM Raleigh

Randall Baartman, Douglas Berg, Joe Bockhold, Erik Daughtrey, Perry Dykes, Kevin Kepros, Li-Fang Lee, Billy Newport, Terry O'Brien, Randy Schnier, Michael Schmitt, Rob Wisniewski
IBM Rochester

Tom Alcott
IBM Costa Mesa

Roger Cundiff, Geoffrey Hambrick, Richard Mayo, Fergus Stewart
IBM Austin

Richard Backhouse, Erik Burckart, John Cammarata, Andy Ivory, Betsy Riggins,
Ying Wang
IBM Durham

Madhu Chetuparambil
IBM Pittsburgh

David Granshaw, Benjamin Hardill, Andrew Schofield
IBM United Kingdom

Mark Endrei, Richard Raszka
IBM Australia

Thomas Hikade
IBM Austria

Elton C. Oliveira
IBM Brazil

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks™ to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- ▶ Use the online **Contact us** review redbook form found at:

ibm.com/redbooks

- ▶ Send your comments in an email to:

redbook@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HZ8 Building 662
P.O. Box 12195
Research Triangle Park, NC 27709-2195

Summary of changes

This section describes the technical changes made in this edition of the book and in previous editions. This edition may also include minor corrections and editorial changes that are not identified.

Summary of Changes
for SG24-6392-00
for WebSphere Application Server V6 Scalability and Performance Handbook
as created or updated on May 25, 2005.

May 2005, First Edition

This revision reflects the addition, deletion, or modification of new and changed information described below.

This redbook is based on the redbook *IBM WebSphere V5.1 Performance, Scalability, and High Availability, WebSphere Handbook Series*, SG24-6198-01. There is, however, a major change in this redbook compared to the previous version: We have removed the High Availability part of the book and moved it into a new redbook entitled *WebSphere Application Server Network Deployment V6: High availability solutions*, SG24-6688. Therefore, you only find information regarding the new High Availability Manager in this book but no explanation as to how to make WebSphere objects highly available using external clustering software, such as IBM HACMP or Tivoli System Automation. For this kind of information, please refer to the redbook.

All chapters have been updated for IBM WebSphere Application Server Network Deployment V6, so the list below only highlights significant enhancements or changes.

New information

- ▶ Information about how Web servers are now handled inside a cell can be found in Chapter 3, “Introduction to topologies” on page 71 or Chapter 6, “Plug-in workload management and failover” on page 227.
- ▶ The new WebSphere High Availability Manager is introduced in Chapter 9, “WebSphere HAManager” on page 465.

Changed information

- ▶ The chapter entitled “Design for scalability” from the previous version of the book has been enhanced and includes now also information about infrastructure planning, sizing, benchmarking, etc. See Chapter 2, “Infrastructure planning and design” on page 39.
- ▶ We have now two chapters on the WebSphere Edge Components: Chapter 4, “Introduction to WebSphere Edge Components” on page 99 which explains the concepts and features of the Edge Components and Chapter 5, “Using IBM WebSphere Edge Components” on page 127 which gives the installation and configuration details.
- ▶ Caching Proxy installation and configuration was added to Chapter 5, “Using IBM WebSphere Edge Components” on page 127.
- ▶ Chapter 7, “EJB workload management” on page 341. Updated for version 6. Information about failover of stateful session beans was added.
- ▶ Chapter 10, “Dynamic caching” on page 501 has new sections, for example 10.3, “The cachespec.xml configuration file” on page 511 and 10.3.2, “Dynamic Cache Policy Editor” on page 515.
- ▶ Chapter 12, “Using and optimizing the default messaging provider” on page 643 explains how the new default messaging provider can be used and optimized for a IBM WebSphere Application Server Network Deployment V6 environment.
- ▶ Chapter 14, “Server-side performance and analysis tools” on page 769 has a new section on application management that gives an introduction to Tivoli Monitoring for Transaction Performance V5.3 (TMTP) and WebSphere Studio Application Monitor V3.1 (WSAM).
- ▶ The Profiler section in Chapter 15, “Development-side performance and analysis tools” on page 839 has been rewritten.
- ▶ A new structure and with it many enhancements has been applied to Chapter 16, “Application development: best practices for application design, performance and scalability” on page 895. This new structure lists the recommendations as they apply to the various layers, for example, the presentation or business logic layer.



Part 1

Getting started

Archived

Archived

Overview and key concepts

This chapter provides a conceptual overview of the goals and issues associated with scaling applications in the WebSphere environment. It also presents the various techniques that can be used to implement scaling.

A short section highlights the performance improvements of WebSphere Application Server V6 over previous versions (see 1.5, “Performance improvements over previous versions” on page 31).

A prerequisite for all WebSphere scalability and workload management concepts discussed in this redbook is the use of IBM WebSphere Application Server Network Deployment V6 or IBM WebSphere Business Integration Server Foundation V6. We assume the use of IBM WebSphere Application Server Network Deployment throughout this book.

1.1 Objectives

As outlined by the product documentation, a typical minimal configuration for a WebSphere Application Server V6 installation is illustrated in Figure 1-1. There is a single Web (HTTP) server and a single application server, and both are co-resident on a single machine. Please note that such a configuration is not recommended for security reasons; there is no DMZ established. This is true for both internal (intranet) and external (Internet) applications. Naturally, the performance characteristics of this setup are limited by the power of the machine on which it is running, by various constraints inherent in the configuration itself, and by the implementation of WebSphere Application Server. A short introduction to the various components of the WebSphere runtime is provided in 1.2, “WebSphere Application Server architecture” on page 12.

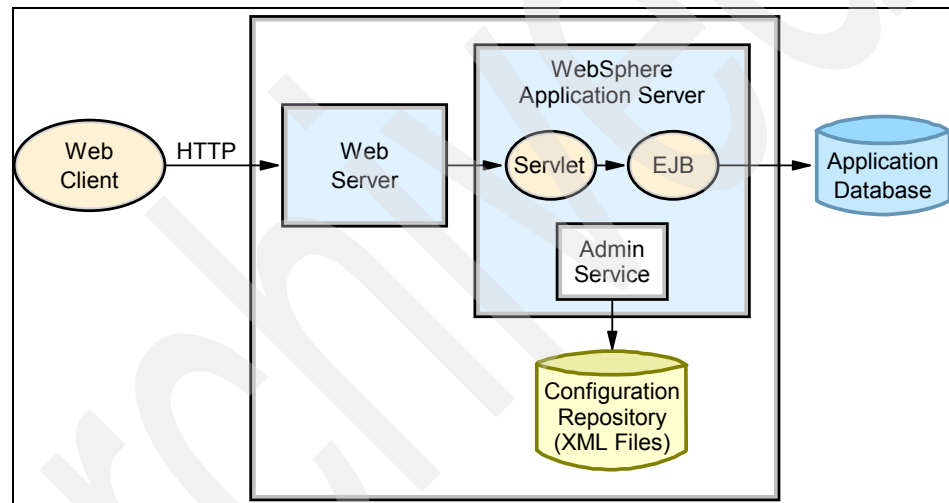


Figure 1-1 Basic WebSphere configuration

The objective of this book is to explore how this basic configuration can be extended to provide more computing power by better exploiting the power of each machine and by using multiple machines. Specifically, we are interested in defining system configurations that exhibit the following properties:

- Scalability
- Workload management
- Availability
- Maintainability
- Session management
- Performance impacts of WebSphere Application Server security

Note that scaling the application server environment does not help if your application has an unscalable design. For Web application and EJB development best practices, refer to Chapter 16, “Application development: best practices for application design, performance and scalability” on page 895 and to the white paper *WebSphere Application Server Development Best Practices for Performance and Scalability*, found at:

http://www.ibm.com/software/webservers/appserv/ws_bestpractices.pdf

1.1.1 Scalability

Scalability defines how easily a site will expand. Web sites must expand, sometimes with little warning, and grow to support an increased load. The increased load may come from many sources:

- ▶ New markets
- ▶ Normal growth
- ▶ Extreme peaks

An application and infrastructure that is architected for good scalability makes site growth possible and easy.

Most often, one achieves scalability by adding hardware resources to improve throughput. A more complex configuration, employing additional hardware, should allow one to service a higher client load than that provided by the simple basic configuration. Ideally, it should be possible to service any given load simply by adding additional servers or machines (or upgrading existing resources).

However, adding new systems or processing power does not always provide a linear increase in throughput. For example, doubling the number of processors in your system will not necessarily result in twice the processing capacity. Nor will adding an additional horizontal server in the Application Server tier necessarily result in twice the request serving capacity. Adding additional resources introduces additional overhead for resource management and request distribution. While the overhead and corresponding degradation may be small, you need to remember that adding n additional machines does not always result in n times the throughput.

Also, you should not simply add hardware without doing some investigation and possible software tuning first to identify potential bottlenecks in your application or any other performance-related software configurations. Adding more hardware may not necessarily improve the performance if the software is badly designed or not tuned correctly. Once the software optimization has been done, then the hardware resources should be considered as the next step for improving performance. See also section 14.2, “Scalability” of the redbook *IBM WebSphere V6 Planning and Design Handbook*, SG24-6446 for information about this topic.

There are two different ways to improve performance when adding hardware: vertical scaling and horizontal scaling. See “Scalability” on page 21 for more information about these concepts.

Performance

Performance involves minimizing the response time for a given request or transaction.

The response time is the time it takes from the moment the user initiates a request at the browser to the moment the result of the HTML page returns to the browser. At one time, there was an unwritten rule of the Internet known as the “8 second rule.” This rule stated that any page that did not respond within eight seconds would be abandoned. Many enterprises still use this as the response time benchmark threshold for Web applications.

Throughput

Throughput, while related to performance, more precisely defines the number of concurrent transactions that can be accommodated.

For example, if an application can handle 10 customer requests simultaneously and each request takes one second to process, this site has a potential throughput of 10 requests per second.

1.1.2 Workload management

Workload management (WLM) is a WebSphere facility to provide load balancing and affinity between application servers in a WebSphere clustered environment. Workload management can be an important facet of performance. WebSphere uses workload management to send requests to alternate members of the cluster. WebSphere also routes concurrent requests from a user to the application server that serviced the first request, as EJB calls, and session state will be in memory of this application server.

The proposed configuration should ensure that each machine or server in the configuration processes a fair share of the overall client load that is being processed by the system as a whole. In other words, it is not efficient to have one machine overloaded while another machine is mostly idle. If all machines have roughly the same capacity (for example, CPU power), each should process a roughly equal share of the load. Otherwise, there likely needs to be a provision for workload to be distributed in proportion to the processing power available on each machine.

Furthermore, if the total load changes over time, the system should automatically adapt itself; for example, all machines may use 50% of their capacity, or all machines may use 100% of their capacity. But not one machine uses 100% of its capacity while the rest uses 15% of their capacity.

In this redbook, we discuss both Web server load balancing using WebSphere Edge Components and WebSphere workload management techniques.

1.1.3 Availability

Also known as resiliency, availability is the description of the system's ability to respond to requests no matter the circumstances. Availability requires that the topology provide some degree of process redundancy in order to eliminate single points of failure. While vertical scalability can provide this by creating multiple processes, the physical machine then becomes a single point of failure. For this reason, a high availability topology typically involves horizontal scaling across multiple machines.

Hardware-based high availability

Using a WebSphere Application Server multiple machine configuration eliminates a given application server process as a single point of failure. In WebSphere Application Server V5.0 and higher, the removal of the application dependencies on the administrative server process for security, naming and transactions further reduces the potential that a single process failure can disrupt processing on a given node. In fact, the only single point of failure in a WebSphere cell is the Deployment Manager, where all central administration is performed. However, a failure at the Deployment Manager only impacts the ability to change the cell configuration and to run the Tivoli Performance Viewer which is now included in the Administrative Console; application servers are more self-sufficient in WebSphere Application Server V5.0 and higher compared to WebSphere Application Server V4.x. A number of alternatives exist to provide high availability for the Deployment Manager, including the possible use of an external high availability solution, though the minimal impact of a Deployment Manager outage typically does not require the use of such a solution. In many cases, manual solutions such as the one outlined in the article "Implementing a Highly Available Infrastructure for WebSphere Application Server Network Deployment, Version 5.0 without Clustering" provides suitable availability. This article is available at

http://www-128.ibm.com/developerworks/websphere/library/techarticles/0304_alcott/alcott.html

See the redbook *WebSphere Application Server Network Deployment V6: High availability solutions*, SG24-6688 for further details on external HA solutions.

Failover

The proposition to have multiple servers (potentially on multiple independent machines) naturally leads to the potential for the system to provide failover. That is, if any one machine or server in the system were to fail for any reason, the system should continue to operate with the remaining servers. The load balancing property should ensure that the client load gets redistributed to the remaining servers, each of which will take on a proportionately slightly higher percentage of the total load. Of course, such an arrangement assumes that the system is designed with some degree of overcapacity, so that the remaining servers are indeed sufficient to process the total expected client load.

Ideally, the failover aspect should be totally transparent to clients of the system. When a server fails, any client that is currently interacting with that server should be automatically redirected to one of the remaining servers, without any interruption of service and without requiring any special action on the part of that client. In practice, however, most failover solutions may not be completely transparent. For example, a client that is currently in the middle of an operation when a server fails may receive an error from that operation, and may be required to retry (at which point the client would be connected to another, still available server). Or the client may observe a pause or delay in processing, before the processing of its requests resumes automatically with a different server. The important point in failover is that each client, and the set of clients as a whole, is able to eventually continue to take advantage of the system and receive service, even if some of the servers fail and become unavailable. Conversely, when a previously failed server is repaired and again becomes available, the system may transparently start using that server again to process a portion of the total client load.

The failover aspect is also sometimes called fault tolerance, in that it allows the system to survive a variety of failures or faults. It should be noted, however, that failover is only one technique in the much broader field of fault tolerance, and that no such technique can make a system 100 percent safe against every possible failure. The goal is to greatly minimize the *probability* of system failure, but keep in mind that the *possibility* of system failure cannot be completely eliminated.

Note that in the context of discussions on failover, the term *server* most often refers to a physical machine (which is typically the type of component that fails). However, we will see that WebSphere also allows for the possibility of one server process on a given machine to fail independently, while other processes on that same machine continue to operate normally.

HAManager

WebSphere V6 introduces a new concept for advanced failover and thus higher availability, called the High Availability Manager (HAManager). The HAManager enhances the availability of WebSphere singleton services like transaction

services or JMS message services. It runs as a service within each application server process that monitors the health of WebSphere clusters. In the event of a server failure, the HAManager will failover the singleton service and recover any in-flight transactions. Please see Chapter 9, “WebSphere HAManager” on page 465 for details.

1.1.4 Maintainability

Maintainability is the ability to keep the system running before, during, and after scheduled maintenance. When considering maintainability in performance and scalability, remember that maintenance periodically needs to be performed on hardware components, operating systems, and software products in addition to the application components.

While maintainability is somewhat related to availability, there are specific issues that need to be considered when deploying a topology that is maintainable. In fact, some maintainability factors are at cross purposes to availability. For instance, ease of maintainability would dictate that one should minimize the number of application server instances in order to facilitate online software upgrades. Taken to the extreme, this would result in a single application server instance, which of course would not provide a high availability solution. In many cases, it is also possible that a single application server instance would not provide the required throughput or performance.

Some of the maintainability aspects that we consider are:

- ▶ Dynamic changes to configuration
- ▶ Mixed configuration
- ▶ Fault isolation

Dynamic changes to configuration

In certain configurations, it may be possible to modify the configuration on the fly without interrupting the operation of the system and its service to clients. For example, it may be possible to add or remove servers to adjust to variations in the total client load. Or it may be possible to temporarily stop one server to change some operational or tuning parameters, then restart it and continue to serve client requests. Such characteristics, when possible, are highly desirable, since they enhance the overall manageability and flexibility of the system.

Mixed configuration

In some configurations, it may be possible to mix multiple versions of a server or application, so as to provide for staged deployment and a smooth upgrade of the overall system from one software or hardware version to another. Coupled with the ability to make dynamic changes to the configuration, this property may be used to effect upgrades without any interruption of service.

Fault isolation

In the simplest application of failover, we are only concerned with clean failures of an individual server, in which a server simply ceases to function completely, but this failure has no effect on the health of other servers. However, there are sometimes situations where one malfunctioning server may in turn create problems for the operation of other, otherwise healthy servers. For example, one malfunctioning server may hoard system resources, or database resources, or hold critical shared objects for extended periods of time, and prevent other servers from getting their fair share of access to these resources or objects. In this context, some configurations may provide a degree of fault isolation, in that they reduce the potential for the failure of one server to affect other servers.

1.1.5 Session state

Unless you have only a single application server or your application is completely stateless, maintaining state between HTTP client requests also plays a factor in determining your configuration. Use of the session information, however, is a fine line between convenience for the developer and performance and scalability of the system. It is not practical to eliminate session data altogether, but care should be taken to minimize the amount of session data passed. Persistence mechanisms decrease the capacity of the overall system, or incur additional costs to increase the capacity or even the number of servers. Therefore, when designing your WebSphere environment, you need to take session needs into account as early as possible.

In WebSphere V6, there are two methods for sharing of sessions between multiple application server processes (cluster members). One method is to persist the session to a database. An alternate approach is to use memory-to-memory session replication functionality, which was added to WebSphere V5 and is implemented using WebSphere internal messaging. The memory-to-memory replication (sometimes also referred to as “in-memory replication”) eliminates a single point of failure found in the session database (if the database itself has not been made highly available using clustering software). See 1.4.1, “HTTP sessions and the session management facility” on page 25 for additional information.

Tip: Refer to the redbook *A Practical Guide to DB2 UDB Data Replication V8*, SG24-6828, for details about DB2 replication.

1.1.6 Performance impact of WebSphere Application Server security

The potential to distribute the processing responsibilities between multiple servers and, in particular, multiple machines, also introduces a number of

opportunities for meeting special security constraints in the system. Different servers or types of servers may be assigned to manipulate different classes of data or perform different types of operations. The interactions between the various servers may be controlled, for example through the use of firewalls, to prevent undesired accesses to data.

SSL

Building up SSL communication causes extra HTTP requests and responses between the machines and every SSL message is encrypted on one side and decrypted on the other side.

SSL at the front end or Web tier is a common implementation. This allows for secured purchases, secure viewing of bank records, etc. SSL handshaking, however, is expensive from a performance perspective. The Web server is responsible for encrypting data sent to the user, and decrypting the request from the user. If a site will be operated using SSL more often than not and the server is operating at close to maximum CPU then using some form of SSL accelerator hardware in the server, or even an external device that offloads all SSL traffic prior to communication with the Web server, may help improve performance.

Performance related security settings

The following settings can help to fine-tune the security-related configurations to enhance performance:

- ▶ **Security cache timeout**

Its setting determines how long WebSphere should cache information related to permission and security credentials. When the cache timeout expires, all cached information becomes invalid. Subsequent requests for the information result in a database lookup. Sometimes, acquiring the information requires invoking an LDAP-bind or native authentication, both of which are relatively costly operations in terms of performance.

- ▶ **HTTP session timeout**

This parameter specifies how long a session will be considered active when it is unused. After the timeout, the session expires and another session object will be created. With high-volume Web sites, this may influence the performance of the server.

- ▶ **Registry and database performance**

Databases and registries used by an application influence the WebSphere Application Server performance. In a distributed environment, it is highly recommended that you put the authorization server onto a separate machine in order to offload application processing. When planning for a secured environment, special attention should be given to the sizing and implementation of the directory server. It is a best practice to first tune the

directory server and make sure that it performs well before looking at the WebSphere environment. Also, make sure that this directory does not introduce a new single point of failure.

Note: WebSphere security is out of the scope of this redbook. However, an entire redbook is dedicated to this topic. See *WebSphere Application Server V6: Security Handbook*, SG24-6316 for details.

1.2 WebSphere Application Server architecture

This section introduces the major components within WebSphere Application Server. Refer to the Redpaper entitled *WebSphere Application Server V6 architecture*, REDP3918 and to the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451 for details about the WebSphere Application Server architecture and components.

1.2.1 WebSphere Application Server Network Deployment components

The following is a short introduction of each WebSphere Network Deployment runtime component and their functions:

- ▶ **Deployment Manager**

The Deployment Manager is part of an IBM WebSphere Application Server Network Deployment V5.x and higher installation. It provides a central point of administrative control for all elements in a distributed WebSphere cell. The Deployment Manager is a required component for any vertical or horizontal scaling and there exists only one specimen of it at a time.

- ▶ **Node Agent**

The Node Agent communicates directly with the Deployment Manager, and is used for configuration synchronization and administrative tasks such as stopping/starting of individual application servers and performance monitoring on the application server node.

- ▶ **Application server (instance)**

An application server in WebSphere is the process that is used to run your servlet and/or EJB-based applications, providing both Web container and EJB container.

- ▶ **Web server and Web server plug-in**

While the Web server is not strictly part of the WebSphere runtime, WebSphere communicates with your Web server of choice: IBM HTTP Server

powered by Apache, Lotus Domino, Microsoft® Internet Information Server, Apache, Sun ONE Web server/Sun Java System Web Server, and Covalent Enterprise Ready Server via a plug-in. This plug-in communicates requests from the Web server to the WebSphere runtime. In WebSphere Application Server V6, the Web server and plug-in XML file can also be managed from the Deployment Manager. For more information about this new function, see 3.4, “Web server topology in a Network Deployment cell” on page 78.

- WebContainer Inbound Chain (formerly embedded HTTP transport)

The WebContainer Inbound Chain is a service of the Web container. An HTTP client connects to a Web server and the HTTP plug-in forwards the requests to the application server through the WebContainer Inbound Chain. The communication type is either HTTP or HTTPS.

Although the WebContainer Inbound Chain is available in any WebSphere Application Server to allow testing or development of the application functionality prior to deployment, it should *not* be used for production environments. See 6.2.1, “WebContainer Inbound Chains” on page 231 for more details.

- Default messaging provider

An embedded JMS server was introduced with WebSphere Application Server V5 but has been totally rewritten for WebSphere Application Server V6 and is now called the default messaging provider. This infrastructure supports both message-oriented and service-oriented applications, and is fully integrated within WebSphere Application Server V6 (all versions). The default messaging provider is a JMS 1.1 compliant JMS provider that supports point-to-point and publish/subscribe styles of messaging.

- Administrative service

The administrative service runs within each WebSphere Application Server Java virtual machine (JVM). Depending on the version installed, there can be administrative services installed in many locations. In WebSphere Application Server V6 and in WebSphere Application Server - Express V6, the administrative service runs in each application server. In a Network Deployment configuration, the Deployment Manager, Node Agent, and application server(s), all host an administrative service. This service provides the ability to update configuration data for the application server and its related components.

- Administrative Console

The WebSphere Administrative Console provides an easy-to-use, graphical “window” to a WebSphere cell and runs in a browser. In a Network Deployment environment, it connects directly to the Deployment Manager, which allows management of all nodes in the cell.

- ▶ Configuration repository

The configuration repository stores the configuration for all WebSphere Application Server components inside a WebSphere cell. The Deployment Manager communicates changes in configuration and runtime state to all cell members through the administrative service and the Node Agent. Each application server maintains a subset of the cell's configuration repository, which holds this application server's configuration information.

- ▶ Administrative cell

A cell is a set of application servers managed by the same Deployment Manager. Some application server instances may reside on the same node, others on different nodes. Cell members do not necessarily run the same application.

- ▶ Server cluster and cluster members

A server cluster is a set of application server processes (the cluster members). They run the same application, but usually on different nodes. The main purpose of clusters is load balancing and failover.

Figure 1-2 on page 15 shows the relationship between these components.

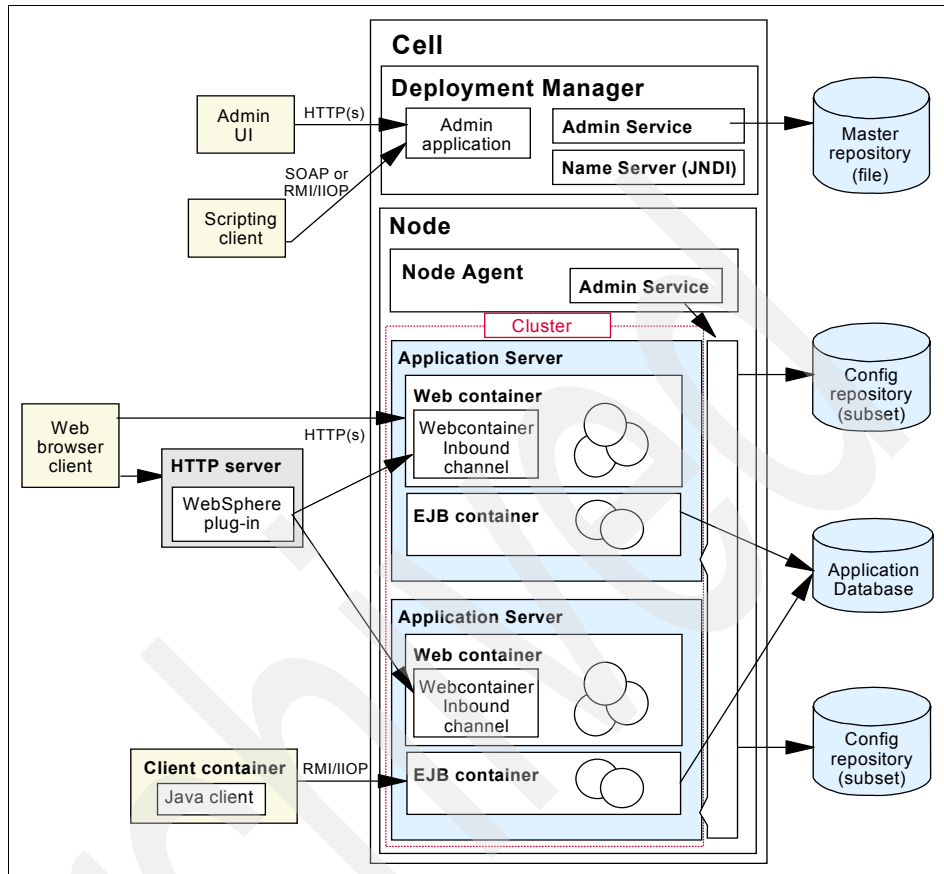


Figure 1-2 WebSphere configuration with Java clients and Web clients

1.2.2 Web clients

The basic configuration shown in Figure 1-1 on page 4 refers to a particular type of client for WebSphere characterized as a Web client. Such a client uses a Web browser to interact with the WebSphere Application Servers, through the intermediary of a Web server (and plug-in), which in turn invokes a servlet within WebSphere (you can also access static HTML pages).

1.2.3 Java clients

Although Web clients constitute the majority of users of WebSphere today, WebSphere also supports another type of client characterized as a Java client. Such a client is a stand-alone Java program (GUI-based or not), which uses the Java RMI/IIOP facilities to make direct method invocations on various EJB objects within an application server, without going through an intervening Web server and servlet, as shown in Figure 1-2 on page 15.

1.3 Workload management

Workload Management (WLM) is the process of spreading multiple requests for work over the resources that can do the work. It optimizes the distribution of processing tasks in the WebSphere Application Server environment. Incoming work requests are distributed to the application servers and other objects that can most effectively process the requests.

Workload management is also a procedure for improving performance, scalability, and reliability of an application. It provides failover when servers are not available.

Workload management is most effective when the deployment topology is comprised of application servers on multiple machines, since such a topology provides both failover and improved scalability. It can also be used to improve scalability in topologies where a system is comprised of multiple servers on a single, high-capacity machine. In either case, it enables the system to make the most effective use of the available computing resources.

Workload management provides the following benefits to WebSphere applications:

- ▶ It balances client processing requests, allowing incoming work requests to be distributed according to a configured WLM selection policy.
- ▶ It provides failover capability by redirecting client requests to a running server when one or more servers are unavailable. This improves the availability of applications and administrative services.
- ▶ It enables systems to be scaled up to serve a higher client load than provided by the basic configuration. With clusters and cluster members, additional instances of servers can easily be added to the configuration. See 1.3.3, “Workload management using WebSphere clustering” on page 19 for details.
- ▶ It enables servers to be transparently maintained and upgraded while applications remain available for users.
- ▶ It centralizes administration of application servers and other objects.

Two types of requests can be workload-managed in IBM WebSphere Application Server Network Deployment V6:

- ▶ *HTTP requests* can be distributed across multiple Web containers, as described in 1.3.2, “Plug-in workload management” on page 17. We refer to this as *Plug-in WLM*.
- ▶ *EJB requests* can be distributed across multiple EJB containers, as described in 1.3.4, “Enterprise Java Services workload management” on page 23. We refer to this as *EJS WLM*.

1.3.1 Web server workload management

If your environment uses multiple Web servers, a mechanism is needed to allow servers to share the load. The HTTP traffic must be spread among a group of servers. These servers must appear as one server to the Web client (browser), making up a *cluster*, which is a group of independent nodes interconnected and working together as a single system. This cluster concept should not be confused with a WebSphere cluster as described in 1.3.3, “Workload management using WebSphere clustering” on page 19.

As shown in Figure 1-3, a load balancing mechanism called *IP spraying* can be used to intercept the HTTP requests and redirect them to the appropriate machine on the cluster, providing scalability, load balancing, and failover.

See Chapter 4, “Introduction to WebSphere Edge Components” on page 99 for details.

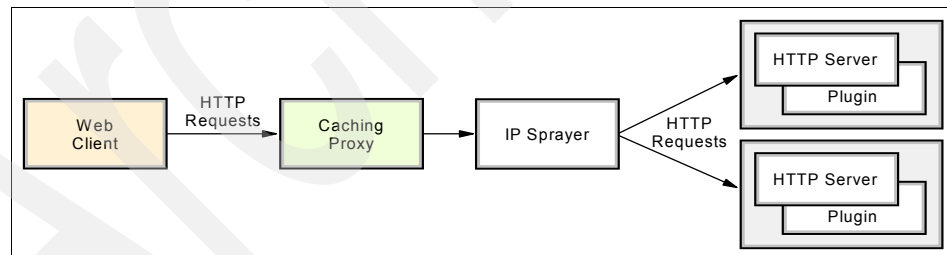


Figure 1-3 Web server workload management

1.3.2 Plug-in workload management

This is a short introduction into plug-in workload management, where servlet requests are distributed to the Web container in clustered application servers, as shown in Figure 1-4 on page 18. This configuration is also referred to as the *servlet clustering architecture*.

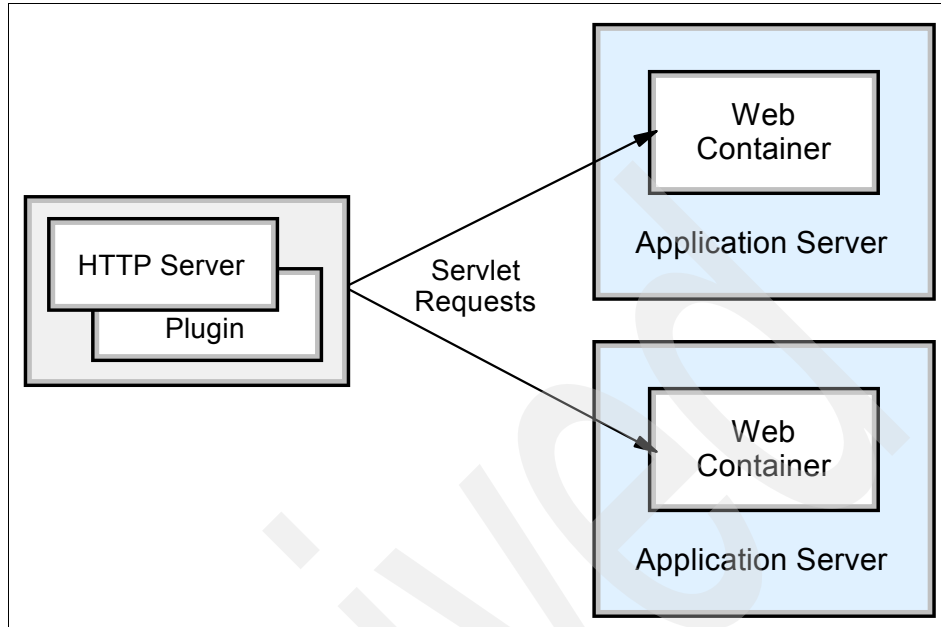


Figure 1-4 Plug-in (Web container) workload management

Clustering application servers that host Web containers automatically enables plug-in workload management for the application servers and the servlets they host. In the simplest case, the cluster is configured on a single machine, where the Web server process also runs.

Routing of servlet requests occurs between the Web server plug-in and the clustered application servers using HTTP or HTTPS. This routing is based purely on the weights associated with the cluster members. If all cluster members have identical weights, the plug-in sends an equal number of requests to all members of the cluster, when assuming no session affinity. If the weights are scaled in the range from zero to 20, the plug-in routes requests to those cluster members with the higher weight value more often. A rule of thumb formula for determining routing preference would be:

$$\% \text{ routed to Server1} = \text{weight1} / (\text{weight1} + \text{weight2} + \dots + \text{weightn})$$

where n is the number of cluster members in the cluster. The Web server plug-in distributes requests around cluster members that are not available.

See Chapter 6, “Plug-in workload management and failover” on page 227 for details.

1.3.3 Workload management using WebSphere clustering

This section describes how workload management is implemented in IBM WebSphere Application Server Network Deployment V6 by using application server clusters and cluster members. How to create clusters and cluster members is detailed in Chapter 8, “Implementing the sample topology” on page 387.

A *cluster* is a set of application servers that are managed together and participate in workload management. Application servers participating in a cluster can be on the same node or on different nodes. A Network Deployment cell can contain no clusters, or have many clusters depending on the need of the administration of the cell.

The cluster is a logical representation of the application servers. It is not necessarily associated with any node, and does not correspond to any real server process running on any node. A cluster contains only application servers, and the weighted workload capacity associated with those servers.

When creating a cluster, it is possible to select an existing application server as the template for the cluster without adding that application server into the new cluster (the chosen application server is used only as a template, and is not affected in any way by the cluster creation). All other *cluster members* are then created based on the configuration of the first cluster member.

Cluster members can be added to a cluster in various ways: during cluster creation and afterwards. During cluster creation, one existing application server can be added to the cluster and/or one or more new application servers can be created and added to the cluster. There is also the possibility of adding additional members to an existing cluster later on. Depending on the capacity of your systems, you can define different weights for the various cluster members.

It may be a good idea to create the cluster with a single member first, adjust the member's configuration and then add the other members. This process guarantees that all cluster members are created with the same settings. Cluster members are required to have identical application components, but they can be sized differently in terms of weight, heap size, and other environmental factors. You must be careful though not to change anything that might result in different application behavior on each cluster member. This concept allows large enterprise machines to belong to a cluster that also contains smaller machines such as Intel® based Windows servers.

Starting or stopping the cluster automatically starts or stops all cluster members, and changes to the application are propagated to all application servers in the cluster.

Figure 1-5 shows an example of a possible configuration that includes server clusters. Server cluster 1 has two cluster members on node B only. Server cluster 2, which is completely independent of server cluster 1, has two cluster members on node A and three cluster members on node B. Finally, node A also contains a free-standing application server that is not a member of any cluster.

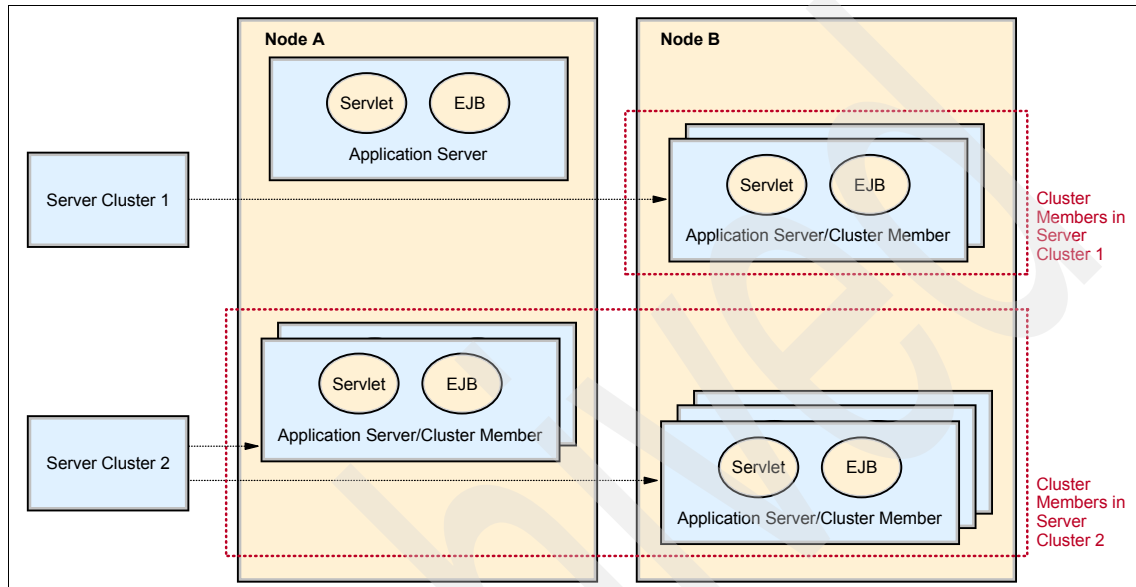


Figure 1-5 Server clusters and cluster members

Clusters and cluster members provide the necessary support for workload management, failover, and scalability. For details, please refer to Chapter 6, “Plug-in workload management and failover” on page 227.

Distributing workloads

The ability to route a request to any server in a group of clustered application servers allows the servers to share work and improving throughput of client requests. Requests can be evenly distributed to servers to prevent workload imbalances in which one or more servers has idle or low activity while others are overburdened. This load balancing activity is a benefit of workload management. Using weighted definitions of cluster members allows nodes to have different hardware resources and still participate in a cluster. The weight specifies that the application server with a higher weight will be more likely to serve the request faster, and workload management will consequently send more requests to that node.

Failover

With several cluster members available to handle requests, it is more likely that failures will not negatively affect throughput and reliability. With cluster members distributed to various nodes, an entire machine can fail without any application downtime. Requests can be routed to other nodes if one node fails. Clustering also allows for maintenance of nodes without stopping application functionality.

Scalability

Clustering is an effective way to perform vertical and horizontal scaling of application servers.

- In *vertical scaling*, shown in Figure 1-6, multiple cluster members for an application server are defined on the same physical machine, or node, which may allow the machine's processing power to be more efficiently allocated.

Even if a single JVM can fully utilize the processing power of the machine, you may still want to have more than one cluster member on the machine for other reasons, such as using vertical clustering for software failover. If a JVM reaches a table/memory limit (or if there is some similar problem), then the presence of another process provides for failover.

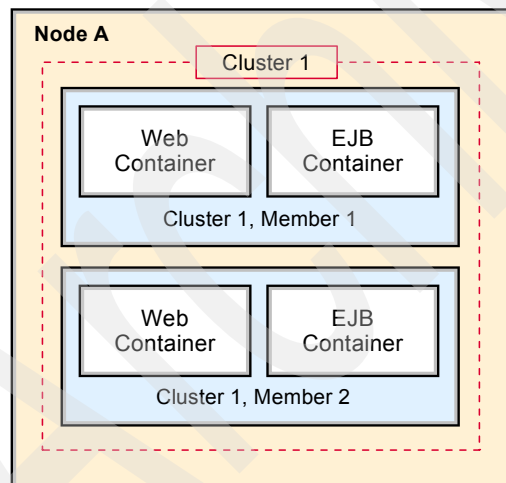


Figure 1-6 Vertical scaling

We recommend that you avoid using “rules of thumb” when determining the number of cluster members needed for a given machine. The only way to determine what is correct for your environment and application(s) is to tune a single instance of an application server for throughput and performance, then

add it to a cluster, and incrementally add additional cluster members. Test performance and throughput as each member is added to the cluster. Always monitor memory usage when you are configuring a vertical scaling topology and do not exceed the available physical memory on a machine.

In general, 85% (or more) utilization of the CPU on a large server shows that there is little, if any, performance benefit to be realized from adding additional cluster members.

- In *horizontal scaling*, shown in Figure 1-7, cluster members are created on multiple physical machines. This allows a single WebSphere application to run on several machines while still presenting a single system image, making the most effective use of the resources of a distributed computing environment. Horizontal scaling is especially effective in environments that contain many smaller, less powerful machines. Client requests that overwhelm a single machine can be distributed over several machines in the system. Failover is another benefit of horizontal scaling. If a machine becomes unavailable, its workload can be routed to other machines containing cluster members.

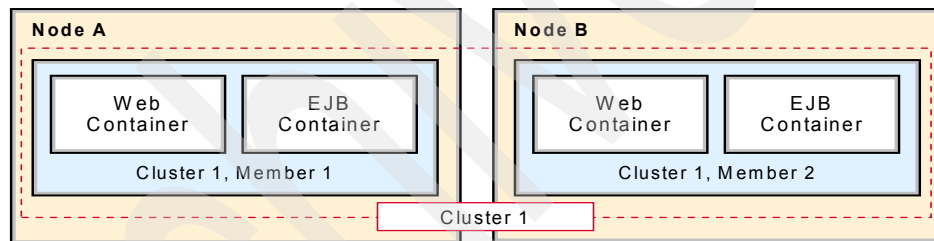


Figure 1-7 Horizontal scaling

Horizontal scaling can handle application server process failures and hardware failures (or maintenance) without significant interruption to client service.

Note: WebSphere Application Server V5.0 and higher supports horizontal clustering across different platforms and operating systems. Horizontal cloning on different platforms was not supported in WebSphere Application Server V4.

- WebSphere applications can combine horizontal and vertical scaling to reap the benefits of both scaling techniques, as shown in Figure 1-8 on page 23.

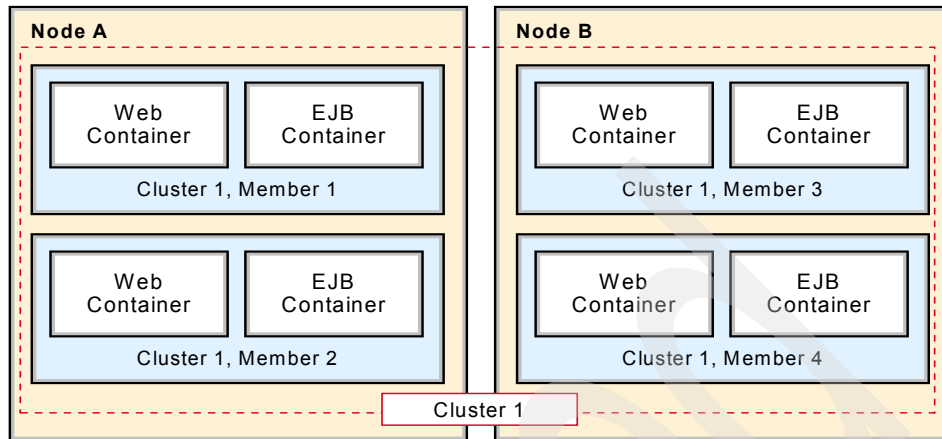


Figure 1-8 Vertical and horizontal scaling

Secure application cluster members

The workload management service has its own built-in security, which works with the WebSphere Application Server security service to protect cluster member resources. If security is needed for your production environment, enable security before you create a cluster for the application server. This enables security for all of the members in that cluster.

The EJB method permissions, Web resource security constraints, and security roles defined in an enterprise application are used to protect EJBs and servlets in the application server cluster. Refer to *WebSphere Application Server V6: Security Handbook*, SG24-6316 for more information.

1.3.4 Enterprise Java Services workload management

In this section, we discuss Enterprise Java Services workload management (EJS WLM), in which the Web container and EJB container are on different application servers. The application server hosting the EJB container is clustered.

Configuring the Web container in a separate application server from the Enterprise JavaBean container (an EJB container handles requests for both session and entity beans) enables distribution of EJB requests between the EJB container clusters, as seen in Figure 1-9 on page 24.

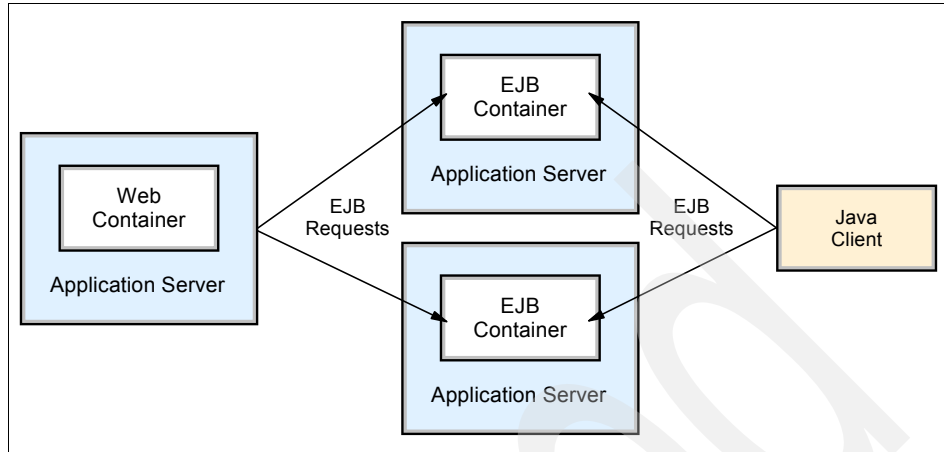


Figure 1-9 EJB workload management

In this configuration, EJB client requests are routed to available EJB containers based on the workload management EJB selection policy (Server-weighted round robin routing or Prefer local).

Important: Although it is possible to split the Web container and EJB container, it is not recommended because of the negative performance impact. In addition, application maintenance also becomes more complex when the application runs in different application servers.

The EJB clients can be servlets operating within a Web container, stand-alone Java programs using RMI/IIOP, or other EJBs.

EJS workload management is covered in detail in Chapter 7, “EJB workload management” on page 341.

1.4 Managing session state among servers

All the load distribution techniques discussed in this book rely, on one level or another, on using multiple copies of an application server and arranging for multiple consecutive requests from various clients to be serviced by different servers.

If each client request is completely independent of every other client request, then it does not matter if two requests are processed on the same server or not. However, in practice, there are many situations where all requests from an individual client are not totally independent. In many usage scenarios, a client makes one request, waits for the result, then makes one or more subsequent requests that depend upon the results received from the earlier requests.

Such a sequence of operations on behalf of one client falls into one of two categories:

- ▶ **Stateless:** the server that processes each request does so based solely on information provided with that request itself, and not on information that it “remembers” from earlier requests. In other words, the server does not need to maintain state information between requests.
- ▶ **Stateful:** the server that processes a request *does* need to access and maintain state information generated during the processing of an earlier request.

Again, in the case of stateless interactions, it does not matter if different requests are being processed by different servers. But in the case of stateful interactions, we must ensure that whichever server is processing a request has access to the state information necessary to service that request. This can be ensured either by arranging for the same server to process all the client requests associated with the same state information, or by arranging for that state information to be shared and equally accessible by all servers that may require it. In that last case, it is often advantageous to arrange for most accesses to the shared state to be performed from the same server, so as to minimize the communications overhead associated with accessing the shared state from multiple servers.

This consideration for the management of stateful interactions is a factor in the discussions of various load distribution techniques throughout this book. It also requires the discussion of a specific facility, the session management facility, and of server affinity.

1.4.1 HTTP sessions and the session management facility

In the case of an HTTP client interacting with a servlet, the state information associated with a series of client requests is represented as an HTTP session, and identified by a session ID. The session manager module that is part of each Web container is responsible for managing HTTP sessions, providing storage for session data, allocating session IDs, and tracking the session ID associated with each client request, through the use of cookies, URL rewriting, or SSL ID techniques.

The session manager provides for the storage of session-related information either in-memory within the application server, in which case it cannot be shared with other application servers, in a back-end database, shared by all application servers, or by using memory-to-memory replication.

The second option, sometimes referred to as *persistent sessions* or *session clustering*, is one method that uses HTTP sessions with the load-distribution configuration. With this option, whenever an application server receives a request associated with a session ID, which is not in memory, it can obtain it by accessing the back-end database, and can then serve the request. When this option is not enabled, and another clustering mechanism is not used, if any load distribution mechanism happens to route an HTTP request to an application server other than the one where the session was originally created, that server would be unable to access the session, and would thus not produce correct results in response to that request. One drawback to the database solution, just as with application data, is that it provides a single point of failure (SPOF) so it should be implemented in conjunction with hardware clustering products such as HACMP or solutions such as database replication. Another drawback is the performance hit, caused by database disk I/O operations and network communications.

The last option, memory-to-memory replication, provides a mechanism for session clustering within IBM WebSphere Application Server Network Deployment. It uses the built-in Data Replication Service (DRS) of WebSphere to replicate session information stored in memory to other members of the cluster. Using this functionality removes the single point of failure that is present in persistent sessions through a database solution that has not been made highly available using clustering software. The sharing of session state is handled by creating a replication domain and then configuring the Web container to use that replication domain to replicate session state information to the specified number of application servers. DRS, like database replication, also incurs a performance hit, primarily because of the overhead of network communications. Additionally, since copies of the session object reside in application server memory this reduces the available heap for application requests and usually results in more frequent garbage collection cycles by the application server JVM.

Note: DRS, and thus the session replication mechanism, has changed in WebSphere V6. Configuration has been greatly simplified. The default configuration setting is now to have a single replica. You can also replicate to the entire domain (which corresponds to the peer-to-peer configuration in WebSphere V5.x) or specify the number of replicas.

Performance measurements showed that the overhead associated with replicating the session to every other cluster member is more significant than contention introduced by using the database session repository (the more application servers in the cluster, the more overhead for memory-to-memory replication). Therefore, it is not recommended that you use the Entire Domain setting for larger replication domains.

For larger session sizes, the overhead of session replication increases. Database replication has a lower overall throughput than memory-to-memory, due to database I/O limitations (the database becomes the bottleneck). However, while database replication with large sessions performs slower, it is doing so with less CPU power than memory-to-memory replication. The unused processor power can be used by other tasks on the system.

Storing session states in a persistent database or using memory-to-memory replication provides a degree of fault tolerance to the system. If an application server crashes or stops, any session state that it may have been working on would normally still be available either in the back-end database or in another still running application server's memory, so that other application servers can take over and continue processing subsequent client requests associated with that session.

Important: Neither mechanism provides a 100% guarantee that a session state will be preserved in case of a server crash. If a server happens to crash while it is literally in the middle of modifying the state of a session, some updates may be lost and subsequent processing using that session may be affected. Also, in the case of memory replication, if the node crashes during a replication, or in between the time interval that has been set for replicating session information, then some session information may be lost.

However, such a situation represents only a very small window of vulnerability and a very small percentage of all occurrences throughout the life of a system in production.

More information can be found in 6.8, “HTTP session management” on page 279.

1.4.2 EJB sessions or transactions

In the case of an EJB client interacting with one or more EJBs, the management of state information associated with a series of client requests is governed by the EJB specification and implemented by the WebSphere EJS container, and depends on the types of EJBs that are the targets of these requests.

Stateless session bean

By definition, when interacting with a stateless session bean, there is no client-visible state associated with the bean. Every client request directed to a stateless session bean is independent of any previous request that was directed to the same bean. The container maintains a pool of instances of stateless session beans of each type, and will provide an arbitrary instance of the appropriate bean type whenever a client request is received. It does not matter if the same actual bean instance is used for consecutive requests, or even if two consecutive requests are serviced by bean instances in the same application server.

Stateful session bean

In contrast, a stateful session bean is used precisely to capture state information that must be shared across multiple consecutive client requests that are part of one logical sequence of operations. The client must take special care to ensure that it is always accessing the same instance of the stateful session bean, by obtaining and keeping an EJB object reference to that bean. The various load-distribution techniques available in WebSphere make special provisions to support this characteristic of stateful session beans.

A new feature in WebSphere V6 is the failover support for stateful session beans; the state information is now replicated to other application servers in the cluster using the Data Replication Service (DRS). If an application server fails, a new instance of the bean is created on a different server, the state information is recovered, requests are directed to the recovered instance and processing continues.

Entity bean

Finally, we must consider the case of an entity bean. Most external clients access WebSphere services through session beans, but it is possible for an external client to access an entity bean directly. Furthermore, a session bean inside WebSphere is itself often acting as a client to one or more entity beans also inside WebSphere; if load distribution features are used between that session bean and its target entity bean, then the same questions arise as with plain external clients.

Strictly speaking, the information contained in an entity bean is not usually associated with a “session” or with the handling of one client request or series of client requests. However, it is common for one client to make a succession of requests targeted at the same entity bean instance. Unlike all the previous cases, it is possible for multiple independent clients to access the same entity bean instance more or less concurrently. Therefore, it is important that the state contained in that entity bean be kept consistent across the multiple client requests.

For entity beans, the notion of a session is more or less replaced by the notion of *transaction*. For the duration of one client transaction to which it participates, the entity bean is instantiated in one container (normally the container where the first operation within that transaction was initiated). All subsequent accesses to that same bean, within that same transaction, must be performed against that same instance in that same container.

In between transactions, the handling of the entity bean is specified by the EJB specification, in the form of a number of caching options:

- ▶ With option A caching, WebSphere Application Server assumes that the entity bean is used within a single container. Clients of that bean must direct their requests to the bean instance within that container. The entity bean has exclusive access to the underlying database, which means that the bean cannot be clustered or participate in workload management if option A caching is used.
- ▶ With option B caching, the bean instance remains active (so it is not guaranteed to be made passive at the end of each transaction), but it is always reloaded from the database at the start of each transaction. A client can attempt to access the bean and start a new transaction on any container that has been configured to host that bean.
- ▶ With option C caching (which is the default), the entity bean is always reloaded from the database at the start of each transaction and made passive at the end of each transaction. A client can attempt to access the bean and start a new transaction on any container that has been configured to host that bean. This is effectively similar to the session clustering facility described for HTTP sessions: the shared state is maintained in a shared database, and can be accessed from any server when required.

Message-driven beans

The message-driven bean (MDB) was introduced with WebSphere V5. Support for MDBs is a requirement of a J2EE 1.3 compliant application server. In WebSphere V4.0, the Enterprise Edition offered a similar functionality called message beans that leveraged stateless session EJBs and a message listener service. That container, however, did not implement the EJB 2.0 specification.

The MDB is a stateless component that is invoked by a J2EE container when a JMS message arrives at a particular JMS destination (either a queue or topic). Loosely, the MDB is triggered by the arrival of a message.

Messages are normally anonymous. If some degree of security is desired, the listener will assume the credentials of the application server process during the invocation of the MDB.

MDBs handle messages from a JMS provider within the scope of a transaction. If transaction handling is specified for a JMS destination, the listener will start a global transaction before reading incoming messages from that destination. Java Transaction API (JTA) transaction control for commit or rollback is invoked when the MDB processing has finished.

1.4.3 Server affinity

The discussion above implies that any load-distribution facility, when it chooses a server to direct a request, is not entirely free to select *any* available server:

- ▶ In the case of stateful session beans or entity beans within the context of a transaction, there is only one valid server. WebSphere WLM will always direct a client's access of a stateful session bean to the single server instance containing the bean (there is no possibility of choosing the wrong server here). If the request is directed to the wrong server (for example because of a configuration error), it will either fail, or that server itself will be forced to forward the request to the correct server, at great performance cost.
- ▶ In the case of clustered HTTP sessions or entity beans between transactions, the underlying shared database ensures that any server can correctly process each request. However, accesses to this underlying database may be expensive, and it may be possible to improve performance by caching the database data at the server level. In such a case, if multiple consecutive requests are directed to the same server, they may find the required data still in the cache, and thereby reduce the overhead of access to the underlying database.

The characteristics of each load-distribution facility, which take these constraints into account, are generally referred to as *server affinity*: In effect, the load distribution facility recognizes that multiple servers may be acceptable targets for a given request, but it also recognizes that each request may have a particular affinity for being directed to a particular server where it may be handled better or faster.

We will encounter this notion of server affinity throughout the discussion of the various load-distribution facilities in Chapter 6, “Plug-in workload management and failover” on page 227 and Chapter 7, “EJB workload management” on page 341. In particular, we will encounter the notion of *session affinity*, where the load distribution facility recognizes the existence of a session and attempts to direct all requests within that session to the same server, and we will also discuss the notion of *transaction affinity*, in which the load distribution facility recognizes the existence of a transaction, and behaves similarly.

Finally, we will also see that a particular server affinity mechanism can be weak or strong. In a weak affinity mechanism, the system attempts to enforce the desired affinity for the majority of requests most of the time, but may not always be able to provide a total guarantee that this affinity will be respected. In a strong affinity mechanism, the system guarantees that affinity will always be strictly respected, and generates an error when it cannot.

1.5 Performance improvements over previous versions

WebSphere Application Server V6 contains a significant set of functional and performance improvements over the initial release of WebSphere Version 5.0 and the incremental releases that followed. Most of the improvements listed below are as compared to WebSphere V5.1.

Improved performance with Java V1.4.2

- ▶ Java Virtual Machine (JVM) enhancements.
- ▶ Various class-level enhancements for logging, BigDecimal, NumberFormat, date formatting and XML parsing.
- ▶ Reduced lock contention for improved ORB scalability.

Improved Web container performance/scalability

- ▶ Channel Framework

The Channel Framework with NIO (a new Java framework for input/output) supports non-blocking IO allowing for a large number of concurrently-connected clients to be supported, which helps client scalability by enabling the WebSphere Application Server containers to handle more client requests with a fewer number of threads than was required previously.

- ▶ Code path improvements

The Web container is redesigned and reimplemented for significant reductions in code path and memory utilization.

- ▶ Caching enhancements

The Web container redesign also contains internal improvements for caching various types of objects inside the Web container. These caching improvements provide better performance and scalability.

- ▶ Improved HTTP session replication using a new high-performing transport

The Domain Replication Service is rebased on the HAManager framework, which uses a multicast/unicast based transport (DCS), which itself uses the channel framework.

See Chapter 6, “Plug-in workload management and failover” on page 227 and Chapter 9, “WebSphere HAManager” on page 465 for more information about this topic.

EJB improvements

- ▶ Cached data verification (functional improvement)

The new cached data verification in the EJB container allows for aggressive caching of data while maintaining consistency with the database by performing an optimistic versioning field check at transaction commit time. This increases performance of transactions and maintains data consistency required in high transaction rate environments.

All EJB applications can benefit from improved EJB performance due to:

- ▶ Code path lengths which are reduced throughout the runtime.

- ▶ Application profiling

The application profiling technology (which was previously only available in IBM WebSphere Application Server Enterprise) improves overall entity EJB performance and throughput by fine tuning the run-time behavior by enabling EJB optimizations to be customized for multiple user access patterns without resorting to “worst case” choices (such as Pessimistic Update on `findByPrimaryKey`, regardless of whether the client needs it for read or for actual update).

- ▶ Higher-performance Access Intent settings

An automatic analysis tool is provided that determines the most performant access intent setting on each entity bean in the application, for each application scenario in which that entity bean may be accessed.

- ▶ Stateful session bean replication

The new stateful session bean replication is using a new high performance transport.

- ▶ Improved persistence manager

The improved persistence manager allows for more caching options and better database access strategies.

Applications with compatible design patterns can exploit the following EJB improvements:

- ▶ Keep-update-locks optimization for DB2 UDB V8.2

This enables applications to use a row blocking protocol for data fetching and also avoid deadlocks by enforcing update locks on affected rows. This feature enhances performance for applications that use search updates, such as EJB CMP Entity Applications.

- ▶ Read-only beans

The new "Read Only Beans" optimization provides enhanced EJB caching performance.

- ▶ Improved read-ahead

Improved Read-ahead functionality enables reading in data from multiple CMP with a single SQL statement.

See Chapter 7, "EJB workload management" on page 341 for more information about this topic.

Improved Web Services performance

Web Services performance improvements can be seen in several areas:

- ▶ Improved deserialization

Improvements in V6.0 tooling generates higher-performing deserializers for all JAX-RPC beans. Because of this, redeploying a V5.x application into V6.0 may significantly decrease the processing time for large messages.

- ▶ Web Services caching

Additional capabilities are added for Web Services caching. The V6.0 serialization/deserialization runtime is enhanced to cache frequently used serializers/deserializers, significantly decreasing the processing time for large messages. Web Services caching also includes JAX/RPC client side Web Services caching (first introduced in V5.1.1).

- ▶ Other Web Services improvements come in the areas of in-process performance enhancements.

Refer to the redbook *WebSphere Version 6 Web Services Handbook Development and Deployment*, SG24-6461 for more information about Web Services.

Default messaging provider

The embedded JMS server from V5.x has been replaced by a new, in-process Java messaging engine that makes nonpersistent messaging primitives up to 5x faster.

The improved default messaging provider for V6.0 is a new all-Java implementation that runs within the WebSphere Application Server JVM process and uses a relational database for its persistent store. It provides improved function and performance over V5.1, particularly in the nonpersistent messaging scenarios. Nonpersistent point-to-point in-process messaging (EJB to MDB) is up to 5 times faster than V5.1. See Chapters 10 and 11 of the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451 for more information.

Improved Dynamic Fragment Caching

DMap caching (or DistributedMap caching), which was previously only available in IBM WebSphere Application Server Enterprise, performs better than command caching because of how the caching mechanisms are implemented.

See Chapter 10, “Dynamic caching” on page 501 for more information.

Type 4 JDBC driver

The Universal JDBC Driver performs better than the legacy/CLI JDBC driver, due mainly to significant code path reductions, a reduced dependency on JNI, and optimizations not available to the legacy/CLI JDBC driver architecture.

Tip: The IBM WebSphere Performance team publishes a performance report for each release of WebSphere Application Server. This performance report provides performance information using the Trade benchmark, information about the new default messaging provider, memory considerations, and much more. To obtain a copy of the WebSphere performance report, please contact your IBM sales or technical support contact.

1.6 The structure of this redbook

This redbook is divided into six parts containing the following chapters and appendixes:

Part 1: Getting started

Chapter 2, “Infrastructure planning and design” on page 39 introduces scalability and scaling techniques, and helps you to evaluate the components of your e-business infrastructure for their scalability. Basic considerations regarding infrastructure deployment planning, sizing, benchmarking, and performance tuning are also covered in this chapter.

Chapter 3, “Introduction to topologies” on page 71 discusses topologies supported by WebSphere Application Server V6 and the variety of factors that come into play when considering the appropriate topology choice.

Part 2: Distributing the workload

Chapter 4, “Introduction to WebSphere Edge Components” on page 99 gives an overview of the capabilities and options of WebSphere Edge Components.

Chapter 5, “Using IBM WebSphere Edge Components” on page 127 looks at Web server load balancing with the WebSphere Edge Components of IBM WebSphere Application Server Network Deployment V6, describing some important configurations and its use with the IBM WebSphere Application Server Network Deployment V6. An introduction to the Caching Proxy is also included in this chapter.

Chapter 6, “Plug-in workload management and failover” on page 227 covers Web container workload management. It discusses components, configuration settings, and resulting behaviors. It also covers session management in a workload-managed environment.

Chapter 7, “EJB workload management” on page 341 examines how EJB requests can be distributed across multiple EJB containers.

Part 3: Implementing the solution

Chapter 8, “Implementing the sample topology” on page 387 provides step-by-step instructions for implementing a sample multi-machine environment. This environment is used to illustrate most of IBM WebSphere Application Server Network Deployment V6’s workload management and scalability features throughout this book, including a Caching Proxy and Load Balancer.

Part 4: High availability and caching

Chapter 9, “WebSphere HAManager” on page 465 introduces the new HAManager feature of WebSphere Application Server V6 and shows some basic usage and configuration examples.

Chapter 10, “Dynamic caching” on page 501 describes the use of caching technologies in three-tier and four-tier application environments involving browser-based clients and applications using WebSphere Application Server.

Part 5: Messaging

Chapter 11, “Using asynchronous messaging for scalability and performance” on page 621 gives an introduction into the JMS 1.1 API.

Chapter 12, “Using and optimizing the default messaging provider” on page 643 introduces the new default messaging provider that is part of WebSphere Application Server V6. It also explains configuration options in a clustered application server environment. This chapter extends Chapters 10 and 11 of the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451 and should thus be read in conjunction with these chapters.

Chapter 13, “Understanding and optimizing the use of WebSphere MQ” on page 697 describes how the various components for WebSphere MQ interact, takes you through manually configuring the components for the Trade 6 application, and demonstrates running this with WebSphere MQ and WebSphere Business Integration Event broker.

Part 6: Performance monitoring, tuning and coding practices

Chapter 14, “Server-side performance and analysis tools” on page 769 explains the tools that can be run on the application server side to analyze and monitor the performance of the environment. The topics covered include the Performance Monitoring Infrastructure, Request Metrics, Tivoli Performance Viewer, and the Performance Advisors.

Chapter 15, “Development-side performance and analysis tools” on page 839 offers an introduction into the tools that come with IBM Rational Application Developer V6.0, such as the profiling tools or the separately downloadable Page Detailer.

Chapter 16, “Application development: best practices for application design, performance and scalability” on page 895 gives suggestions for developing WebSphere Application Server based applications.

Chapter 17, “Performance tuning” on page 939 discusses tuning an existing environment for performance and provides a guide for testing application servers

and components of the total serving environment that should be examined and potentially tuned to increase the performance of the application

Part 6: Appendixes

Appendix A, “Sample URL rewrite servlet” on page 1033 explains how to set up an example to test session management using URL rewrites.

Appendix B, “Additional material” on page 1037 gives directions for downloading the Web material associated with this redbook.



Infrastructure planning and design

Are you wondering about how to plan and design an infrastructure deployment based on WebSphere middleware? This chapter covers the WebSphere-specific components that you need to be familiar with to run a successful WebSphere infrastructure project.

This chapter is organized in the following sections:

- ▶ Infrastructure deployment planning
- ▶ Design for scalability
- ▶ Sizing
- ▶ Benchmarking
- ▶ Performance tuning

2.1 Infrastructure deployment planning

This section gives a general overview of which phases you have to go through during a project, how you gather requirements, and how to apply them to a WebSphere project.

Typically, a new project starts with only a concept. Very little is known about specific implementation details, especially as they relate to the infrastructure. Hopefully, your development team and infrastructure team work closely together to help bring some scope to the needs of the overall application environment.

In some cases, the involvement of an IBM Design Center for e-business on demand™ might be useful. For more information about this service, please see 2.1.1, “IBM Design Centers for e-business on demand” on page 41.

Bringing together a large team of people can create an environment that helps hone the environment requirements. If unfocused, however, a large team can be prone to wandering aimlessly and creating more confusion than resolving issues. For this reason, pay close attention to the size of the requirements team, and keep the meetings focused. Provide template documents to be completed by the developers, the application business owners, and the user experience team. Try to gather information that falls into the following categories:

- ▶ Functional requirements, which are usually determined by the business-use of the application and are related to function.
- ▶ Non-functional requirements, which start to describe the properties of the underlying architecture and infrastructure like reliability, availability, or security.
- ▶ Capacity requirements, which include traffic estimates, traffic patterns, and expected audience size.

Requirement gathering is an iterative process. There is no way, especially in the case of Web-based applications, to have absolutes for every item. The best you can do is create an environment that serves your best estimates, and then monitor closely to adjust as necessary after launch. Make sure that all your plans are flexible enough to deal with future changes in requirements and always keep in mind that they may have an impact on every other part of your project.

With this list of requirements, you can start to create the first drafts of your designs. You should target developing at least the following designs:

- ▶ Application design

To create your application design, you use your functional and non-functional requirements to create guidelines for your application developers about how your application is built.

This redbook does not attempt to cover the specifics of a software development cycle. There are multiple methodologies for application design, and volumes dedicated to best practices. However, Chapter 16, “Application development: best practices for application design, performance and scalability” on page 895 will give you a quick start to this topic.

► Implementation design

This design defines your target deployment infrastructure on which your application will be deployed.

The final version of this implementation design will contain every detail about hardware, processors, and software installed on the different components, but you do not start out with all these details in the beginning. Initially, your implementation design will simply list component requirements such as a database, a set of application servers, a set of Web servers, and whatever other components are defined in the requirements phase.

This design might need to be extended during your project, whenever a requirement for change occurs, or when you get new sizing information. Too often, however, the reality is that a project may require new hardware, and therefore be constrained by capital acquisition requirements. Try to not go back too often for additional resources once the project has been accepted.

Information that helps you to create this design is found in 2.2, “Design for scalability” on page 42.

With the two draft designs in hand, you can now begin the process of formulating counts of servers, network requirements, and the other infrastructure related items. Basic information about sizing can be found in 2.3, “Sizing” on page 58.

In some cases, it might be appropriate to perform benchmark tests. There are many ways to perform benchmarking tests, and some of these methods are described in 2.4, “Benchmarking” on page 59.

The last step in every deployment is to tune your system and measure if it can handle the projected load specified in your non-functional requirements. Section 2.5, “Performance tuning” on page 62 covers in more detail how to plan for load tests.

2.1.1 IBM Design Centers for e-business on demand

The IBM Design Centers for e-business on demand are state-of-the-art facilities where certified IT architects work with customers from around the world to design, architect, and prototype advanced IT infrastructure solutions to meet the challenges of on demand computing. Customers are nominated by their IBM representative based upon their current e-business plans and are usually designing a first-of-a-kind e-business infrastructure to solve a unique business

problem. After a technical and business review, qualified customers come to a Design Center for a Design Workshop which typically lasts three to five days. If appropriate, a proof-of-concept session ranging from two to eight weeks may follow.

The centers are located in:

- ▶ Poughkeepsie, USA
- ▶ Montpellier, France
- ▶ Makuhari, Japan

For more information about this service, please go to

http://www-1.ibm.com/servers/eserver/design_center/

2.2 Design for scalability

Understanding the scalability of the components in your e-business infrastructure and applying appropriate scaling techniques can greatly improve availability and performance. Scaling techniques are especially useful in multi-tier architectures, when you want to evaluate components associated with IP load balancers such as dispatchers or edge servers, Web presentation servers, Web application servers, data servers and transaction servers.

You can use the following high-level steps to classify your Web site and identify scaling techniques that are applicable to your environment:

1. Understanding the application environment
2. Categorizing your workload
3. Determining the most affected components
4. Selecting the scaling techniques to apply
5. Applying the technique(s)
6. Re-evaluating

This systematic approach needs to be adapted to your situation. We look at each step in detail in the following sections.

2.2.1 Understanding the application environment

For existing environments, the first step is to identify all components and understand how they relate to each other. The most important task is to understand the requirements and flow of the existing application(s) and what can or cannot be altered. The application is a significant contributor to the scalability of any infrastructure, so a detailed understanding is crucial to scale effectively. At a minimum, application analysis must include a breakdown of transaction types and volumes as well as a graphic view of the components in each tier.

Figure 2-1 can aid in determining where to focus scalability planning and tuning efforts. The figure shows the greatest latency for representative customers in three workload patterns, and on which tier the effort should be concentrated.

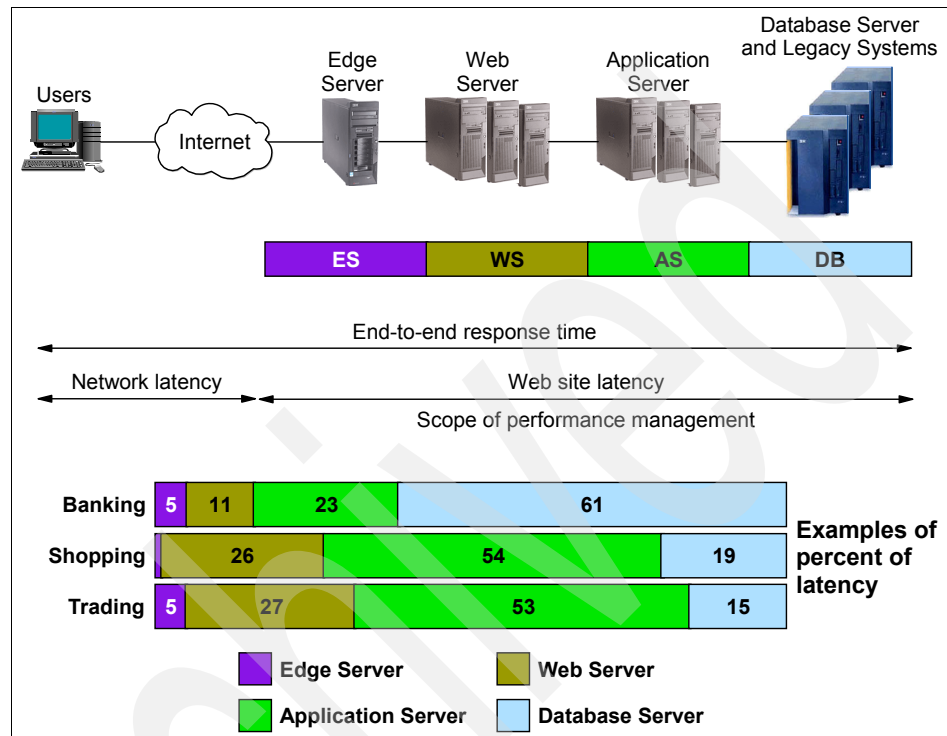


Figure 2-1 How latency varies based on workload pattern and tier

For example, for online banking, most of the latency typically occurs in the database server, whereas the application server typically experiences the greatest latency for online shopping and trading sites. Planning to resolve those latencies from an infrastructure solution becomes very different as the latency shifts.

The way applications manage traffic between tiers significantly affects the distribution of latencies between the tiers, which suggests that careful analysis of application architectures is an important part of this step and could lead to reduced resource utilization and faster response times. Collect metrics for each tier, and make user-behavior predictions for each change implemented. WebSphere offers various facilities for monitoring the performance of an application, as discussed in Chapter 14, “Server-side performance and analysis tools” on page 769 and Chapter 15, “Development-side performance and analysis tools” on page 839.

As requirements are analyzed for a new application, strive to build scaling techniques into the infrastructure. Implementation of new applications offer the opportunity to consider each component type, such as open interfaces and new devices, and the potential to achieve unprecedented transaction rates. Each scaling technique affects application design. Similarly, application design impacts the effectiveness of the scaling technique. To achieve proper scale, application design must consider potential architectural scaling effects. An iterative, incremental approach will need to be followed in the absence of known workload patterns.

2.2.2 Categorizing your workload

Knowing the workload pattern for a site determines where to focus scalability efforts and which scaling techniques to apply. For example, a customer self-service site such as an online bank needs to focus on transaction performance and the scalability of databases that contain customer information used across sessions. These considerations would not typically be significant for a publish/subscribe site, where a user signs up for data to be sent to them, usually via a mail message.

Workload patterns and Web site classifications

Web sites with similar workload patterns can be classified into site types. Consider five distinct workload patterns and corresponding Web site classifications:

- ▶ Publish/subscribe (user to data)
- ▶ Online shopping (user to online buying)
- ▶ Customer self-service (user to business)
- ▶ Online trading (user to business)
- ▶ Business to business

Publish/subscribe (user to data)

Sample publish/subscribe sites include search engines, media sites such as newspapers and magazines, as well as event sites such as sports championships (for example the site for the Olympics).

Site content changes frequently, driving changes to page layouts. While search traffic is low in volume, the number of unique items sought is high, resulting in the largest number of page views of all site types. Security considerations are minor compared to other site types. Data volatility is low. This site type processes the fewest transactions and has little or no connection to legacy systems. Refer to Table 2-1 on page 45 for details on these workload patterns.

Table 2-1 Publish/subscribe site workload pattern

Workload pattern	Publish/subscribe
Categories/examples	<ul style="list-style-type: none"> ▶ Search engines ▶ Media ▶ Events
Content	<ul style="list-style-type: none"> ▶ Dynamic change of the layout of a page, based on changes in content, or need ▶ Many page authors; page layout changes frequently ▶ High volume, non-user-specific access ▶ Fairly static information sources
Security	Low
Percent secure pages	Low
Cross-session info	No
Searches	<ul style="list-style-type: none"> ▶ Structured by category ▶ Totally dynamic ▶ Low volume
Unique Items	High
Data volatility	Low
Volume of transactions	Low
Legacy integration/ complexity	Low
Page views	High to very high

Online shopping (user to online buying)

Sample sites include typical retail sites where users buy books, clothes, or even cars. Site content can be relatively static, such as a parts catalog, or dynamic, where items are frequently added and deleted, such as for promotions and special discounts that come and go. Search traffic is heavier than on a publish/subscribe site, although the number of unique items sought is not as large. Data volatility is low. Transaction traffic is moderate to high, and almost always grows.

When users buy, security requirements become significant and include privacy, non-repudiation, integrity, authentication, and regulations. Shopping sites have more connections to legacy systems, such as fulfillment systems, than

publish/subscribe sites, but generally fewer than the other site types. This is detailed in the following table.

Table 2-2 Online shopping site workload pattern

Workload pattern	Online shopping
Categories/examples	<ul style="list-style-type: none"> ▶ Exact inventory ▶ Inexact inventory
Content	<ul style="list-style-type: none"> ▶ Catalog either flat (parts catalog) or dynamic (items change frequently, near real time) ▶ Few page authors and page layout changes less frequently ▶ User-specific information: user profiles with data mining
Security	<ul style="list-style-type: none"> ▶ Privacy ▶ Non-repudiation ▶ Integrity ▶ Authentication ▶ Regulations
Percent secure pages	Medium
Cross-session info	High
Searches	<ul style="list-style-type: none"> ▶ Structured by category ▶ Totally dynamic ▶ High volume
Unique items	Low to medium
Data volatility	Low
Volume of transactions	Moderate to high
Legacy integration/ complexity	Medium
Page views	Moderate to high

Customer self-service (user to business)

Sample sites include those used for home banking, tracking packages, and making travel arrangements. Home banking customers typically review their balances, transfer funds, and pay bills. Data comes largely from legacy applications and often comes from multiple sources, thereby exposing data consistency.

Security considerations are significant for home banking and purchasing travel services, less so for other uses. Search traffic is low volume; transaction traffic is

moderate, but growing rapidly. Refer to Table 2-3 for details on these workload patterns.

Table 2-3 Customer self-service site workload pattern

Workload pattern	Customer self-service
Categories/examples	<ul style="list-style-type: none"> ▶ Home banking ▶ Package tracking ▶ Travel arrangements
Content	<ul style="list-style-type: none"> ▶ Data is in legacy applications ▶ Multiple data sources, requirement for consistency
Security	<ul style="list-style-type: none"> ▶ Privacy ▶ Non-repudiation ▶ Integrity ▶ Authentication ▶ Regulations
Percent secure pages	Medium
Cross-session info	Yes
Searches	<ul style="list-style-type: none"> ▶ Structured by category ▶ Low volume
Unique items	Low
Data volatility	Low
Volume of transactions	Moderate and growing
Legacy integration/ complexity	High
Page views	Moderate to low

Online trading (user to business)

Of all site types, trading sites have the most volatile content, the potential for the highest transaction volumes (with significant swing), the most complex transactions, and are extremely time sensitive. Auction sites are characterized by highly dynamic bidding against items with predictable life times.

Trading sites are tightly connected to the legacy systems. Nearly all transactions interact with the back-end servers. Security considerations are high, equivalent

to online shopping, with an even larger number of secure pages. Search traffic is low volume. Refer to Table 2-4 for details on these workload patterns.

Table 2-4 Online trading site workload pattern

Workload pattern	Online trading
Categories/examples	<ul style="list-style-type: none"> ▶ Online stock trading ▶ Auctions
Content	<ul style="list-style-type: none"> ▶ Extremely time sensitive ▶ High volatility ▶ Multiple suppliers, multiple consumers ▶ Transactions are complex and interact with back end
Security	<ul style="list-style-type: none"> ▶ Privacy ▶ Non-repudiation ▶ Integrity ▶ Authentication ▶ Regulations
Percent secure pages	High
Cross-session info	Yes
Searches	<ul style="list-style-type: none"> ▶ Structured by category ▶ Low volume
Unique items	Low to medium
Data volatility	High
Volume of transactions	High to very high (very large swings in volume)
Legacy integration/ complexity	High
Page views	Moderate to high

Business to business

These sites include dynamic programmatic links between arm's length businesses (where a trading partner agreement might be appropriate). One business is able to discover another business with which it may want to initiate transactions.

In supply chain management, for example, data comes largely from legacy applications and often from multiple sources, thereby exposing data consistency. Security requirements are equivalent to online shopping. Transaction volume is moderate, but growing; transactions are typically complex, connecting multiple

suppliers and distributors. Refer to Table 2-5 for details on these workload patterns.

Table 2-5 Business-to-business site workload pattern

Workload pattern	Business to business
Categories/examples	e-Procurement
Content	<ul style="list-style-type: none">▶ Data is in legacy applications▶ Multiple data sources, requirement for consistency▶ Transactions are complex
Security	<ul style="list-style-type: none">▶ Privacy▶ Non-repudiation▶ Integrity▶ Authentication▶ Regulations
Percent secure pages	Medium
Cross-session info	Yes
Searches	<ul style="list-style-type: none">▶ Structured by category▶ Low to moderate volume
Unique items	Moderate
Data volatility	Moderate
Volume of transactions	Moderate to low
Legacy integration/ complexity	High
Page views	Moderate

Workload characteristics

Your site type will become clear as it is evaluated using Table 2-6 on page 50 for other characteristics related to transaction complexity, volume swings, data volatility, security, and so on.

If the application has further characteristics that could potentially affect its scalability, then add the extra characteristics to Table 2-6 on page 50.

Note: All site types are considered to have high volumes of dynamic transactions. This may or may not be a standard consideration when performing sizing and site classification.

Table 2-6 Workload characteristics and Web site classifications

Workload characteristics	Publish/ subscribe	Online shopping	Customer self- service	Online trading	Business to business	Your workload
Volume of user-specific responses	Low	Low	Medium	High	Medium	
Amount of cross-session information	Low	High	High	High	High	
Volume of dynamic searches	Low	High	Low	Low	Medium	
Transaction complexity	Low	Medium	High	High	High	
Transaction volume swing	Low	Medium	Medium	High	High	
Data volatility	Low	Low	Low	High	Medium	
Number of unique items	High	Medium	Low	Medium	Medium	
Number of page views	High	Medium	Low	Medium	Medium	
Percent secure pages (privacy)	Low	Medium	Medium	High	High	
Use of security (authentication, integrity, non-repudiation)	Low	High	High	High	High	
Other characteristics						High

2.2.3 Determining the most affected components

This step involves mapping the most important site characteristics to each component. Once again, from a scalability viewpoint, the key components of the infrastructure are the load balancers, the Web application servers, security services, transaction and data servers, and the network. Table 2-7 on page 51 specifies the significance of each workload characteristic to each component. As seen in the table, the effect on each component is different for each workload characteristic.

Table 2-7 Load impacts on components

Workload characteristics	Web present. server	Web app. server	Network	Security servers	Fire-walls	Existing business servers	Data-base server
High % user-specific responses	Low	High	Low	Low	Low	High	High
High % cross-session information	Med	High	Low	Low	Low	Low	Med
High volume of dynamic searches	High	High	Med	Low	Med	Med	High
High transaction complexity	Low	High	Low	Med	Low	High	High
High transaction volume swing	Med	High	Low	Low	Low	High	High
High data volatility	High	High	Low	Low	Low	Med	High
High # unique items	Low	Med	Low	Low	Low	High	High
High # page views	High	Low	High	Low	High	Low	Low
High % secure pages (privacy)	High	Low	Low	High	Low	Low	Low
High security	High	High	Low	High	Low	High	Low

2.2.4 Selecting the scaling techniques to apply

The best efforts in collecting the information needed are worthwhile in order to make the best possible scaling decisions. Only when the information gathering is as complete as it can be is it time to consider matching scaling techniques to components.

Manageability, security, and availability are critical factors in all design decisions. Techniques that provide scalability but compromise any of these critical factors cannot be used.

Here is a list of the eight scaling techniques:

- ▶ Using a faster machine
- ▶ Creating a cluster of machines
- ▶ Using appliance servers
- ▶ Segmenting the workload

- ▶ Batch requests
- ▶ Aggregating user data
- ▶ Managing connections
- ▶ Caching

Note: Rather than buying hardware that can handle exponential growth that may or may not occur, consider specific approaches for these three types of servers:

- ▶ For application servers, the main technique for the growth path is to add more machines. It is therefore appropriate to start with the expectation of more than one application server. Depending on the topology, as discussed in Chapter 3, “Introduction to topologies” on page 71, you may assume a load balancer in front of the Web tier, or possibly in front of the application server tier. Adding more machines then becomes easier and far less disruptive.
- ▶ For data servers, get a server that is initially oversized; some customers run at just 30% capacity. This avoids the problem in some environments where the whole site can only use one data server. Another scaling option when more capacity is needed is to partition the database into multiple servers. Another possibility is to partition the data into multiple smaller servers, taking advantage of distributed parallel processing techniques.
- ▶ Just as many sites separate the application server from the database server, so do they separate the Web server from the application server. The front-end Web serving and commerce application functions are placed on less expensive commodity machines. Because these machines are lower cost, a load balancer is also generally deployed with these machines.

Using a faster machine

The goal is to increase the ability to do more work in a unit of time by processing tasks more rapidly. Upgrading the hardware or software will result in a faster machine. However, one of the issues is that software capabilities can limit the hardware exploitation and vice versa. Another issue is that due to hardware or software changes, changes may be needed to existing system management policies.

Creating a cluster of machines

The goal is to service more client requests. Parallelism in machine clusters typically leads to improvements in response time. Database partitioning can be utilized to implement more machines running the database, allowing for parallel processing of large queries. Also, system availability is improved due to failover safety in replicas.

The service running in a replica may have associated with it state information that must be preserved across client requests, and thus should be shared among machines. State sharing is probably the most important issue with machine clusters and can complicate the deployment of this technique. WebSphere's workload balancing feature uses an efficient data-sharing technique to support clustering. Issues such as additional system management for hardware and software can also be challenging.

WebSphere Application Server V6 - Network Deployment provides the WebSphere Edge Components that can be used for Web server load balanced clusters. The clustering concept is discussed in various chapters throughout this book. Details can also be found in Chapter 9, "Topologies" of the redbook *IBM WebSphere V6 Planning and Design Handbook*, SG24-6446, which covers several configurations utilizing clusters to improve performance and scalability.

Using appliance servers

The goal is to improve the efficiency of a specific component by using a special purpose machine to perform the required action. These machines tend to be dedicated machines that are very fast and optimized for a specific function. Examples are network appliances and routers with cache.

Some issues to consider regarding special machines are the sufficiency and stability of the functions and the potential benefits in relation to the added complexity and manageability challenges. Make sure to apply the same resiliency techniques to these devices. Do not introduce an appliance that creates a new single point of failure in the environment.

Segmenting the workload

The goal is to split up the workload into manageable chunks, thereby obtaining a more consistent and predictable response time. The technique also makes it easier to manage which servers the workload is being placed on. Combining segmentation with replication often offers the added benefit of an easy mechanism to redistribute work and scale selectively, as business needs dictate.

An issue with this technique is that, in order to implement the segmentation, the different workloads serviced by the component need to be characterized. Care should be taken during implementation to ease the characterization of the workload. After segmenting the workload, additional infrastructure may be required to balance physical workload among the segments. If you decide that workload segmentation is appropriate for your application, you may want to investigate the use of WebSphere Partitioning Facility (WPF), which is available in WebSphere Extended Deployment. Refer to Chapter 3, "Introduction to topologies" on page 71 and Chapter 9, "Topologies" of the redbook *IBM*

WebSphere V6 Planning and Design Handbook, SG24-6446 for further details on workload segmentation and balancing configurations.

Batch requests

The goal is to reduce the number of requests sent between requesters and responders (such as between tiers or processes) by allowing the requester to define new requests that combine multiple requests.

The benefits of this technique arise from the reduced load on the responders by eliminating overhead associated with multiple requests. It also reduces the latency experienced by the requester due to the elimination of the overhead costs with multiple requests. These requests can be processed offline or at night time to reduce the load during peak hours.

One issue may be limits in achieving reuse of requests due to inherent differences in various requests types (for instance, Web front end differs from voice response front end). This can lead to increased costs of supporting different request types. Another issue is the latency associated with batch applications. There may also be regulations against delay of delivery of that information that restrict the ability to batch requests.

Aggregating user data

The goal is to allow rapid access to large amounts of customer data controlled by existing system applications and support personalization based on customer specific data.

Accessing existing customer data spread across multiple legacy system applications may cause these applications to become overloaded, especially when the access is frequent. This can degrade response time. To alleviate this problem, the technique calls for aggregating customer data into a customer information service (CIS). A CIS that is kept current can provide rapid access to the customer data for a very large number of customers; thus, it can provide the required scalability. Another possible tool is the use of a front end software that maps all the back end data as a single store, though in native format. A product like DB2 Information Integrator provides such a facility.

An issue with a CIS is that it needs to scale very well to support large data as well as to field requests from a large number of application servers (requesters). Data currency issues also can result from creation of a CIS. An issue associated with DB2 Information Integrator type tools is that not all proprietary formats for data are supported. It may be necessary to develop your own maps, depending on the nature of the legacy application.

Managing connections

The goal is to minimize the number of connections needed for an end-to-end system, as well as to eliminate the overhead of setting up connections during normal operations. To reduce the overhead associated with establishing connections between each layer, a pool of pre-established connections is maintained and shared among multiple requests flowing between the layers.

For instance, WebSphere Application Server provides database connection managers to allow connection reuse. It is important to note that a session may use multiple connections to accomplish its tasks, or many sessions may share the same connection. This is called connection pooling in the WebSphere connection manager.

Another pool of connections is at the Web container level. The application server environment pre-allocates this pool to allow for quick access to requests. Once a request has been satisfied, the container releases the connection back to the pool. There are configuration options for allowing this pool to grow beyond the maximum, and to limit how far beyond that it can grow. These, however, can create additional memory requirements. Care should be taken when adjusting the parameters related to the Web container thread pool.

The key issue is with maintaining a session's identity when several sessions share a connection. Reusing existing database connections conserves resources and reduces latency for application requests, thereby helping to increase the number of concurrent requests that can be processed. Managing connections properly can improve scalability and response time. Administrators must monitor and manage resources proactively to optimize component allocation and use.

Refer to Chapter 14, “Server-side performance and analysis tools” on page 769 and Chapter 17, “Performance tuning” on page 939 for more information.

Caching

The goal is to improve performance and scalability by reducing the length of the path traversed by a request and the resulting response, and by reducing the consumption of resources by components.

Caching techniques can be applied to both static and dynamic Web pages. A powerful technique to improve performance of dynamic content is to asynchronously identify and generate Web pages that are affected by changes to the underlying data. Once these changed pages are generated, they must be effectively cached for subsequent data requests. There are several examples of intelligent caching technologies that can significantly enhance the scalability of e-business systems. For static Web pages, the key is to determine the

appropriate content expiration, so that if a site changes, the caching component knows to refresh the cached copy.

The key issue with caching dynamic Web pages is determining what pages should be cached and when a cached page has become obsolete. Information regarding caching techniques can be found in 4.8, “Caching Proxy overview” on page 122 and Chapter 10, “Dynamic caching” on page 501.

At the database layer, caching should be used based on what facilities are provided by the Database Management System. Using data pre-fetching, you can have the database read additional data with the expectation that this data will also be needed very soon. This can increase performance by minimizing the number of disk reads required. Also, memory buffers can be used to house data pages once read into memory. This also will reduce disk access. The key is to make sure the system has adequate main physical memory to provide to the database.

2.2.5 Applying the technique(s)

When performing any scaling, tuning, or adjustments to the application environment, you should first try to apply the selected technique(s) to a test environment to avoid directly impacting the production environment. The idea is to evaluate not only the performance and scalability impact to a component, but also to determine how each change affects the surrounding components and the end-to-end infrastructure. A situation where improvements on one component result in an increased (and unnoticed) load on another component is undesirable.

Figure 2-2 on page 57 illustrates the typical relationship between the techniques and the key infrastructure components. By using this figure, the key techniques for each component can be identified.

In many cases, all techniques cannot be applied because one or more of the following statements are true:

- ▶ Investing in the techniques, even if it would prove helpful, is not affordable.
- ▶ There is no perceived need to scale as much as the techniques will allow.
- ▶ Cost/benefit analysis shows that the technique will not result in a reasonable payback.

Therefore, there must be a process for applying these techniques in different situations so that the best return is achieved. This mapping is a starting point and shows the components to focus on first, based on application workload.

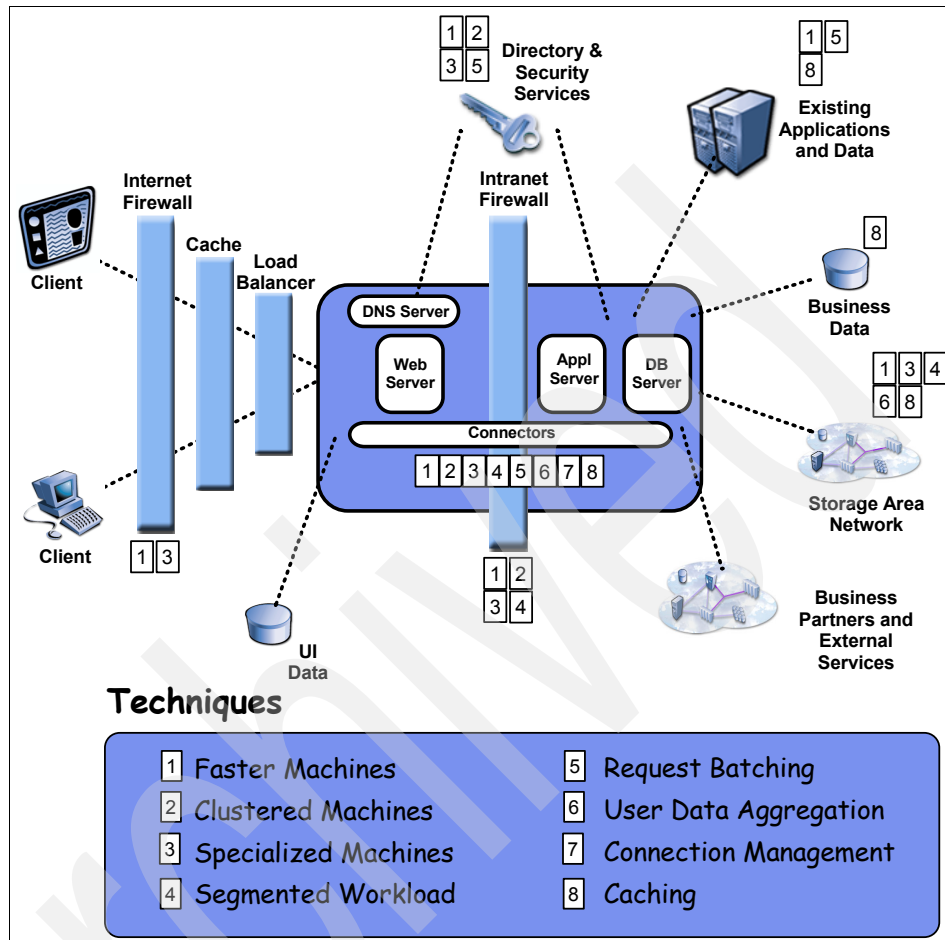


Figure 2-2 Scaling techniques applied to components

2.2.6 Re-evaluating

As with all performance-related work, tuning will be required. The goals are to eliminate bottlenecks, scale to a manageable status for those that cannot be eliminated, and work around those that cannot be scaled. Some examples of tuning actions and the benefits realized are:

- ▶ Increasing Web server threads raises the number of requests that could be processed in parallel.
- ▶ Adding indexes to the data server reduces I/O bottlenecks.
- ▶ Changing the defaults for several operating system variables allows threaded applications to use more heap space.

- ▶ Caching significantly reduces the number of requests to the data server.
- ▶ Increasing the number of edge/appliance servers improves load balancing.
- ▶ Upgrading the data server increases throughput.

Such results demonstrate the potential benefits of systematically applying scaling techniques and continuing to tune.

For an introduction into WebSphere tuning, refer to Chapter 17, “Performance tuning” on page 939.

Recognize that any system is dynamic. The initial infrastructure will, at some point, need to be reviewed, and possibly grown. Changes in the nature of the workload can create a need to re-evaluate the current environment. Large increases in traffic will require examination of the machine configurations. As long as you understand that scalability is not a one-time design consideration, that instead it is part of the growth of the environment, you will be able to keep a system resilient to changes and avoid possibly negative experiences due to poorly planned infrastructure.

For more details on the approach used in this section refer to the Design for Scalability article available at the following URL:

<http://www.ibm.com/developerworks/websphere/library/techarticles/hvws/scalability.html>

2.3 Sizing

After creating a draft of your initial design, you should have built some scalability into the architecture, but you still need to know the number of machines you will have to work with for the project. The sizing of your hardware environment is usually done in cooperation with IBM or a business partner.

For our further discussion, we assume that you do not have your application completed yet because, for most projects, this will be the case. As an example, the overall project budget has to be approved before the development project is started. Or, the developers do not anticipate having the application ready for some time.

It is imperative that you have as static a version of your application design at this point as possible. Work closely with your development team, if at all possible. The better you understand the application, the better your sizing estimation is going to be.

At this point in time, you should also consider which hardware platform you want to deploy and whether you prefer either scaling-up or scaling-out. The hardware platform decision is primarily dependent on your platform preference, which platforms have sizing information available, and which platforms are supported by WebSphere Application Server. Hardware decisions may also be driven by availability of hardware that forces a limitation in the operating systems that can be deployed.

The choice between scaling-up or scaling-out is usually a decision of preference and cost for your environment. The reality is, however, that application resiliency issues may change your preferences. Scaling-up means to implement vertical scaling on a small number of machines with many processors. This can present fairly significant single points of failure (SPOF) because your environment is composed of fewer large machines. Scaling-out, on the other hand, means using a larger number of smaller machines. This can generally be thought of as significantly more resilient, since it is unlikely that the failure of one small server is adequate to create a complete application outage. However, scaling-out introduces a higher maintenance overhead.

What you need to understand, however, is that the sizing estimations are solely based on your input (which means that the better the input, the better the result). The sizing work assumes an average application performance behavior. There are no tests being run, but an average response time is assumed for each transaction and calculations are performed to determine the estimated number of machines and processors your application will require. If your enterprise has a user experience team, they may have documented standards for typical response times that your new project will be required to meet.

If you need a more accurate estimation of your hardware requirements and your application is in a state that allows it to be used for benchmark tests, then you might want to consider one of the benchmarking services offered by IBM or Business Partners as discussed in 2.4, “Benchmarking” on page 59.

Please notice that this sizing is for your production environment only. Based on this estimation, you not only have to update your production implementation design but you also have to update the designs for the integration and development environments accordingly. The key is to remember that changes to the production environment should be incorporated into the development and testing environments as well, if cost considerations do not merit otherwise.

2.4 Benchmarking

Benchmarking is the process used to take an application environment and determine the capacity of that environment through load testing. This allows you

to make reasonable judgements as your environment begins to change. Using benchmarks, you can determine the current work environment capacity, and set expectations as new applications and components are introduced.

Benchmarking is primarily interesting to two kinds of clients:

- ▶ Clients who already have an application and want to migrate to a new version of WebSphere or want to evaluate the exact number of machines for their target deployment platform.
- ▶ Clients who sell products based on WebSphere and want to provide sizing estimations for their products.

Many sophisticated enterprises maintain a benchmark of their application stack and change it after each launch or upgrade of a component. These customers usually have well developed application testing environments, and teams dedicated to the cause. For those that do not, alternatives are available, such as the IBM @server Benchmark Centers or the IBM Test Center. Please refer to “IBM eServer™ Benchmarking Centers” on page 60 and “IBM Test Center” on page 61 for more information about these options.

There are also third party benchmark companies that provide this service. When choosing, make sure that the team performing the benchmark tests has adequate knowledge of the environment, and a clear set of goals. This helps to reduce the costs of the benchmark tests, and creates results that are much easier to quantify.

2.4.1 IBM eServer™ Benchmarking Centers

Learning by trial and error or placing your leading-edge solution into production and then learning it doesn't work the way you expected can be a very painful experience. That's why the IBM @server Benchmark Centers are the resource to tap when you want to minimize risks before moving a new solution into your production environment. This service is available both for customers and IBM Business Partners.

If you need to test your solution in a simulated production environment but you do not have a test environment big enough to stress the solution, the IBM @server Benchmark Centers can run your applications and your configuration and stress the solution for you using the safety of the simulated environment created in the Benchmark Center. You can also use the Benchmark Centers when you require more performance and scalability proof than what's published in industry-standard and application-specific benchmarks. The Benchmark Centers offer the assurance that your developed solution will hold up under the stresses of real-world e-business transactions in your environment. Fees vary by engagement.

To get started, ask your IBM Sales Representative or Business Partner about getting help for testing your solution or to perform a proof of concept in a simulated production environment.

For xSeries®, there is a center in Montpellier, France.

For iSeries, the centers are in:

- ▶ Montpellier, France
- ▶ Rochester, USA

See <http://www-1.ibm.com/servers/eserver/series/benchmark/cbc/> for details.

For pSeries, the centers are in:

- ▶ Montpellier, France
- ▶ Poughkeepsie, USA

For zSeries®, there are centers in:

- ▶ Montpellier, France
- ▶ Boeblingen, Germany
- ▶ Poughkeepsie, USA
- ▶ Gaithersburg, USA

Note: This list of Benchmark Centers might not be complete. If in doubt, please contact your local IBM representative to find centers close to you.

For more information, go to:

<http://www-1.ibm.com/servers/eserver/benchmark.html>

http://www-1.ibm.com/partnerworld/pwhome.nsf/weblook/tech_support_emea_ats.html

IBM Business Partners can also get in touch with an IBM Innovation Center for benchmarking. You will find additional information at:

http://www.developer.ibm.com/en_US/spc/istc/benchmark.html

2.4.2 IBM Test Center

IBM Global Services offers customers the ability to retain IBM for Performance Management, Testing and Scalability services. This team will come to a customer site and assess the overall site performance. This investigation is platform neutral, with no sales team poised to sell additional hardware as a result of the analysis. Offerings include, but are not limited to:

- ▶ Testing and Scalability Services for TCP/IP networks
- ▶ Testing and Scalability Services for Web site stress analysis
- ▶ Performance Engineering and Test Process Consulting
- ▶ Performance Testing and Validation
- ▶ Performance Tuning and Capacity Optimization

When utilizing a service like those provided by the IBM Test Center, you are presented with large amounts of supporting documentation, and evidence to support the results of the tests. This data can then be used to revise your current architecture, or possibly just change the overall infrastructure footprint to add additional machines, or correct single points of failure.

These offerings can be purchased through your local IBM account representative.

2.5 Performance tuning

Performance is one of the most important non-functional requirements for any WebSphere environment. Application performance should be tracked continuously during your project.

Imagine your project is finished; you switch your environment to production, and your environment is unable to handle the user load. This is by far the most user-visible problem that you could have. Most users are willing to accept small functional problems when a system is rolled out, but performance problems are unacceptable to most users and affect everyone working on the system.

2.5.1 Application design problems

Many performance problems, however, cannot be fixed by utilizing more hardware or changing WebSphere parameters as they are related to the application design. Because of this, you really want to make performance testing (and tuning) part of your development and release cycles. It will take much more effort and money to correct the problem after it occurred in production than to fix the problem up front.

The value of frequent testing starting early during the development cycle ensures that performance and scalability bottlenecks are identified early enough to take corrective action either through application remediation or by identifying additional hardware requirements. In addition to minimizing performance and scalability issues, application problems are also identified earlier and have less of an opportunity to manifest themselves in production.

2.5.2 Understand your requirements

Without a clear understanding of your requirements, you have no target to tune against. The most important thing when doing performance tuning is to know your objectives. Do not waste time trying to do performance tuning on a system that was improperly sized and cannot withstand the load no matter how long you tune it. Nor should you continue tuning your system when you are already beyond your performance targets.

You need to know the following two things to understand your requirements:

- ▶ Non-functional requirements
- ▶ Sizing

If you do not have this information and you are asked to tune a system, either you will fail or you will not know when to stop tuning.

2.5.3 Test environment setup

When performing any performance tests, make sure these criteria are applied throughout all your tests:

- ▶ Perform your tests on machines that mirror the production server state. Be as realistic as possible with the test environment. Try to get the closest hardware configuration possible to the production environment. Make sure you will be able to extrapolate the test results to the production environment.
- ▶ Make sure you are using the same data set for all your tests. This allows for better comparability of your test results. At least one of your test environments should also include production data volumes to ensure that all data access paths are realistically tested.
- ▶ Make sure that nobody is using the test machines and that no background processes are running that consume more resources than what you find in production. As an example, if the intent is to test performance during the database backup, then make sure the backup is running.

This means that running your monitoring software in the background, which also runs in production, like Tivoli or BMC is all right, but having a database backup running may not be valid for your desired test.

- ▶ Check CPU, memory and disk utilization before and after each test run to see if there are any unusual patterns. If the target environment will be using shared infrastructure (messaging servers, authentication providers, for example), try to make sure the shared component is performing under a projected shared load.
- ▶ Isolate network traffic as much as possible. Using switches, there is rarely a circumstance where traffic from one server will overrun the port of another. It

is possible, however, to flood ports used for routing off the network to other networks, or even the switch backbone for very heavy traffic. Make sure that your network is designed in such a manner that it isolates the testing environment as much as possible prior to starting, since performance degradation of the network can create unexpected results.

2.5.4 Test phases

There are different test phases during your project cycle. These phases are:

- ▶ Development testing
- ▶ Application and environment testing

Development testing

During the application development phase, the development team should publish regular code base versions and builds to a test server. This activity supports incremental code development, integration, and testing. The development team should also create a Build Verification Test process, one where each new build is executed before making this build available to the team. A Build Verification Test covers one or more test cases or scenarios, and it should test the critical paths through the code that are operational and require validation. Executing the Build Verification Test will ensure that the new build has no obvious errors.

The Build Verification Test process should start as soon as one or more use cases are ready and have been tested on the test server.

The back-end test system where the Build Verification Test is carried out should reasonably mimic the interfaces and communication systems of the production systems to ensure the code can be tested using end-to-end scenarios. The better able the development team is to create an accurate test environment, the more effective and thorough its tests will be.

Application test phase

Following the application development phase is the application test phase. This phase is entirely dedicated to testing and adds value by ensuring that all code is tested at the site level before the site itself becomes fully operational. Unless otherwise indicated, all stress and performance tests should be performed on the actual site code.

The main activities during the application test phase are:

- ▶ Functional Verification Test (FVT)

The main objective of this activity is to test the functionality of the code from end to end, using a suite of test cases.

- ▶ **System Integration Test (SIT)**

This activity is a preparatory phase that determines if the customer test environment is ready to support system tests and User Acceptance Test.

- ▶ **System Verification Test (SVT)**

The purpose of this activity is to validate the functionality of site code received from the development team. All system tests should be performed in the simulated environment.

- ▶ **User Acceptance Test (UAT)**

This activity focuses on the final site look and feel, the site flows and behaviors, and overall business process validation. Normally, the final consumer, the customer, is the one who carries out this activity.

Each activity focuses on a different aspect of the application testing phase; the details of these activities are beyond the scope of this book.

System integration and verification testing

An integration system should be a very close approximation of the logical layout of the production environment. This could mean multiple partitioned machines to simulate the numbers of production machines, or it could mean smaller machines in terms of processing, but equal in number. The following are general recommendations that you may want to follow:

- ▶ For a production system with up to two machines in a single layer, you should either get the same machine for integration testing, or half the processors in each machine.
- ▶ If you have more than two clustered machines in a single layer, you might want to get half the number of machines than you have in production.

It is important to keep in mind that the integration system should be at least half as large as your production environment to get meaningful results, and you want to have the same WLM characteristic that you have in production. This is necessary because WebSphere Application Server behaves differently in a cluster than with a single application server. If using persistent session state, this is an essential item for testing. Again, try to be as realistic as possible, ensuring that you can extrapolate the test results to the production environment.

During the production phase of your application, you use this environment to perform load tests and compare their results to baseline results that you captured with your earlier software builds. Based on this, because your production environment is some multiple of the size of your integration environment, you will be able to tell how the new release impacts your production environment and whether this release is already fit to be deployed there.

2.5.5 Load factors

The most important factors that determine how you conduct your load tests are the following:

- ▶ Request rate
- ▶ Concurrent users
- ▶ Usage patterns

This is not a complete list and other factors may become more important depending on the kind of site being developed. These factors are explained in greater detail in Chapter 17, “Performance tuning” on page 939.

Usage patterns

At this point in the project, it is very important that you think about how your users will use the site. You may want to use the use cases that your developers defined for their application design as the input to build your usage patterns. This makes it easier to later build the scenarios that the load test would use.

Usage patterns consist of the following parts:

- ▶ Use cases modeled as click streams through your pages
- ▶ Weights applied to your use cases

Combining weights with click streams is very important because it shows you how many users you expect in which of your application components and where they generate load.

After all, it is a different kind of load if you expect 70% of your users to search your site instead of browsing through the catalog than the other way around. These assumptions also have an impact on your caching strategy.

To use this information later when recording your load test scenarios, we recommend you write a report with screen shots or URL paths for the click streams (user behavior). Include the weights for your use cases to show the reviewers how the load was distributed.

Make sure that you notify the developers of your findings so they can apply them to their development effort. Also make sure that the most common use cases are the ones where most of the performance optimization work is done.

2.5.6 Production system tuning

This is the only environment that impacts the user experience for your customers. This is where you apply all the performance, scalability and high availability considerations and techniques described throughout this book and where you

tune WebSphere parameters as described in 17.5, “Performance tuning guidelines” on page 975.

Tuning this system is an iterative process and instead of testing multiple application versions here and comparing them to each other, you are changing WebSphere configuration parameters themselves and optimize them to suit your runtime environment.

Important: You should use the final application code to perform tuning in the production environment. This version should have passed performance tests on the integration environment prior to changing any WebSphere parameters on the production system.

When changing a production environment, you should use regular system administration practices, such as those outlined in Chapter 17, “Performance tuning” on page 939. This implies some standard practices:

- ▶ Change only one parameter at a time
- ▶ Document all your changes
- ▶ Compare several test runs to the baseline

These test runs should not differ by more than a small percentage or you have some other problem with your environment that you need to sort out before you continue tuning.

As soon as you have finished tuning your production systems, you should apply the settings, where it makes sense, to your other test environments to make sure they are similar to production. You should also re-run your tests there to establish new baselines on these systems and to see how these changes affect the performance.

Please keep in mind that usually you only have one chance on getting this right. Normally, as soon as you are in production with your system, you can no longer run performance tests on this environment because you cannot take the production system offline to run more performance tests. If a production system is being tested, it is likely that the system is running in a severely degraded position, and you have already lost half the battle.

Note: Because it is rare to use a production system for load tests, it is usually a bad idea to migrate these environments to new WebSphere versions without doing a proper test on an equivalent test system or new hardware.

After completing your first performance tests on your production systems and tuning the WebSphere parameters, you should evaluate your results and compare them to your objectives to see how all of this worked out.

2.5.7 Conclusions

There are various possible outcomes of your performance tests that you should clearly understand and act upon:

- ▶ Performance meets your objectives.

Congratulations! But do not stop here. Make sure you have planned for future growth and are meeting all your performance goals. We recommend documenting your findings in a performance tuning report and archiving it. Include all the settings you changed to reach your objectives.

This report will be very useful when you set up a new environment or you have to duplicate your results somewhere else on a similar environment with the same application. This data will also be essential when adding additional replicas of some component in the system because it ensures that you can change settings on the new system to the settings of the current system.

- ▶ Performance is slower than required.

Your application performance is somewhat slower than expected and you have already performed all possible application and WebSphere parameter tuning. You might need to add more hardware (for example, increase memory, upgrade processors, etc.) to those components in your environment that proved to be bottlenecks during your performance tests. Then run the tests again. Verify with the appropriate teams that there were no missed bottlenecks in the overall system flow.

- ▶ Performance is significantly slower than required.

In this case, you should start over with your sizing and ask the following questions:

- Did you underestimate any of the application characteristics during your initial sizing? If so, why?
- Did you underestimate the traffic and number of users/hits on the site?
- Is it still possible to change parts of the application to improve performance?
- Is it possible to obtain additional resources?

After answering these questions, you should have a better understanding of the problem that you face right now. Your best bet is to analyze your application and try to find the bottlenecks that cause your performance problems. Tools like the Profiler, which is part of IBM Rational Application

Developer V6.0, can help you with this (refer to 15.2, “The Profiler (profiling tools)” on page 840 for more information).

Archived

Introduction to topologies

This chapter discusses some basic topologies and a variety of factors that come into play when considering the appropriate choice, such as:

- ▶ Performance and throughput
- ▶ Scalability, availability, maintainability
- ▶ Security, session state

We describe some topologies that address these factors.

The following topics are discussed in this chapter:

- ▶ J2EE tiers model
- ▶ Topology selection criteria
- ▶ Strategies for scalability
- ▶ Web server topology in a Network Deployment cell
- ▶ Single machine, single node, Web server separated
- ▶ Vertical scaling topology
- ▶ Horizontal scaling topology
- ▶ Horizontal scaling with IP sprayer topology
- ▶ Topology with redundancy of several components
- ▶ The sample topology
- ▶ Topologies and high availability
- ▶ Topology selection summary

3.1 J2EE tiers model

This section covers the basics of a generic J2EE multi-tier application.

Java 2 Platform Enterprise Edition (J2EE) is a set of standards for developing and deploying enterprise Java applications, providing a multi-tier distributed application model, which can be divided basically into four main architectural tiers, as depicted in Figure 3-1:

- ▶ Client tier
- ▶ Presentation tier
- ▶ Business logic tier
- ▶ The Enterprise Information Systems (EIS) tier

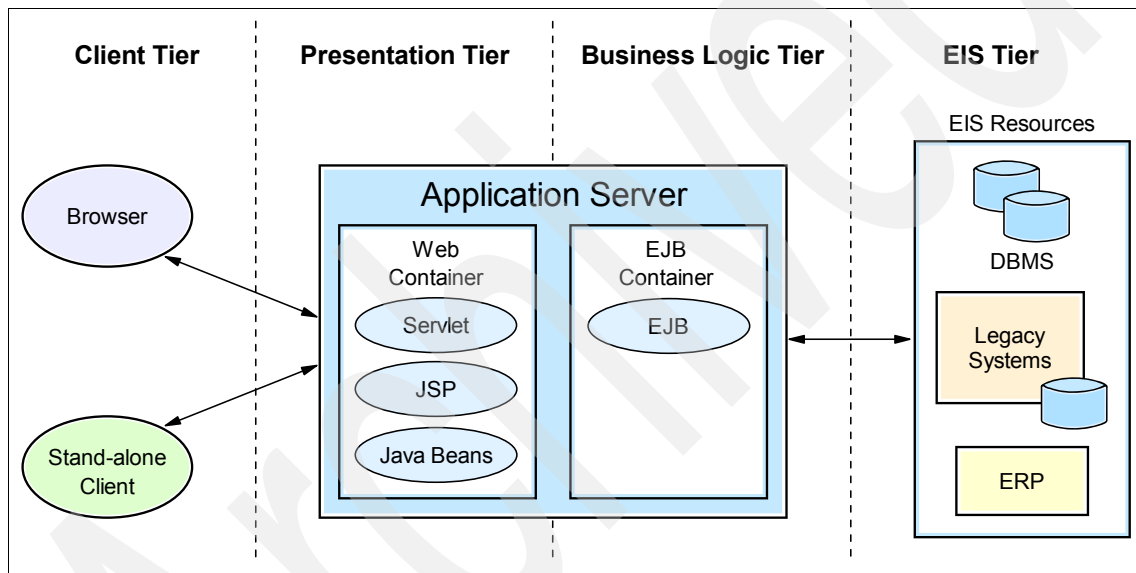


Figure 3-1 Multi-tier application model

The client tier encompasses various client types, such as browsers, applets or stand-alone application clients. These clients can reside both within and outside of the enterprise firewall. User actions are translated into server requests and the server responses are translated into a user-readable format.

The presentation tier embraces Web components, either JSPs or servlets, which are deployed on Web containers. They access data and services from other tiers, handle user requests and screen flow, and can also control user interaction, presenting the returned information back to the client.

Business logic components access enterprise data and business rules, consisting of enterprise beans, deployed on EJB containers. There are three kinds of enterprise beans: session beans, entity beans, and message-driven beans.

The Enterprise Information Systems tier is commonly referred to as the back-end tier; examples include database management systems, mainframe transaction processing and other legacy systems.

J2EE does not specify the structure and implementation of the runtime. It introduces the concept of container; the contract between applications and the container is specified via the J2EE APIs. WebSphere Application Server delivers the infrastructure for deploying J2EE-compliant applications, providing the application servers on which we will run our Java programs. The application server implements the Web container and EJB container components.

On a WebSphere topology, the basic components that interact to execute our application are:

- ▶ The HTTP server and WebSphere Web server plug-in
- ▶ WebSphere Application Server (Web containers and EJB containers)
- ▶ Databases (application databases)

Note: The Web container in WebSphere Application Server has an embedded HTTP transport (the so-called WebContainer Inbound Chain), which allows for direct connection to the application without the need for a separate Web server. While using this transport as a Web server is very handy for testing or development purposes it should not be used in production environments. For performance and security reasons, it is recommended that you use a stand-alone Web server and the HTTP plug-in for the Web server in a production environment.

The emphasis of our topologies scenarios will be the mapping of the J2EE application architecture tiers to a physical deployment on WebSphere Application Server as well as how to apply different techniques and component associations in order to provide scalability, load balancing and failover, based on multiple criteria.

3.2 Topology selection criteria

While a variety of factors come into play when considering the appropriate topology for a WebSphere deployment (see Chapter 1, “Overview and key concepts” on page 3), the primary factors to plan for typically include:

- ▶ Security
- ▶ Performance
- ▶ Throughput
- ▶ Scalability
- ▶ Availability
- ▶ Maintainability
- ▶ Session state

To assist you with selecting a topology, we provide a selection criteria table in 3.12, “Topology selection summary” on page 94.

For detailed information about topologies, their advantages and disadvantages, required software, as well as topology selection criteria, please refer to the redbook *IBM WebSphere V6 Planning and Design Handbook*, SG24-6446.

Note: For each of the Network Deployment topologies, a decision needs to be made regarding the placement of the Deployment Manager and master cell repository. The Deployment Manager can be located either on a dedicated machine, or on the same machine as one of its nodes. It is, however, considered a best practice to place the Deployment Manager on a separate machine. For more information about possible configurations please refer to 9.3, “Cell topologies” in the redbook *IBM WebSphere V6 Planning and Design Handbook*, SG24-6446.

3.3 Strategies for scalability

On demand computing requires the ability to scale up or scale down an application, depending on the current requirements. Thus, scalability is important to improve efficiency and reduce cost.

We start by discussing scalability strategies using WebSphere Application Server that can help us in ensuring high availability, load balancing, and removing bottlenecks.

The basic infrastructure components that make up a WebSphere application are:

- ▶ HTTP server and Web server plug-in
- ▶ Web container
- ▶ EJB container
- ▶ Database(s)

IBM WebSphere Application Server implements Web containers and EJB containers in each application server. The application servers each run in their own JVM (Java Virtual Machine).

If we have all components co-located on a single machine, they might:

- ▶ Compete for machine resources
- ▶ Influence each other's behavior
- ▶ Have unauthorized access to strategic resources

One strategy is to physically separate some components, preventing them from competing for resources (CPU, memory, I/O, network, and so on) or to restrict the access to a resource from another machine (for example, inserting a firewall in order to protect confidential information). This approach is represented in Figure 3-2.

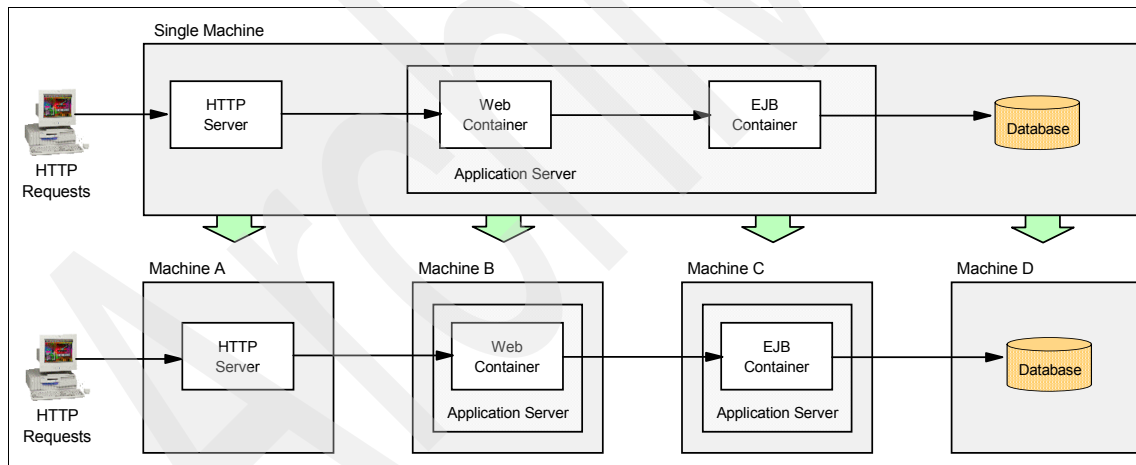


Figure 3-2 Separating components across multiple machines

Important: Figure 3-2 shows the Web container and the EJB container separated on different machines. Although this is a possible configuration, it is not recommended because doing so results in out-of-process calls from the EJB clients in the Web container to the EJB container and will likely have a negative impact on performance. As a result, we will not cover this split JVM topology further in this chapter.

We can also exploit another strategy, distributing the load among the most appropriate resources, and using workload management techniques such as vertical and horizontal scaling, as described in 1.3.3, “Workload management using WebSphere clustering” on page 19. WebSphere Application Servers can benefit from vertical and horizontal scaling and the HTTP servers can be horizontally scaled on a clustered configuration. The use of these techniques is represented in Figure 3-3 on page 77.

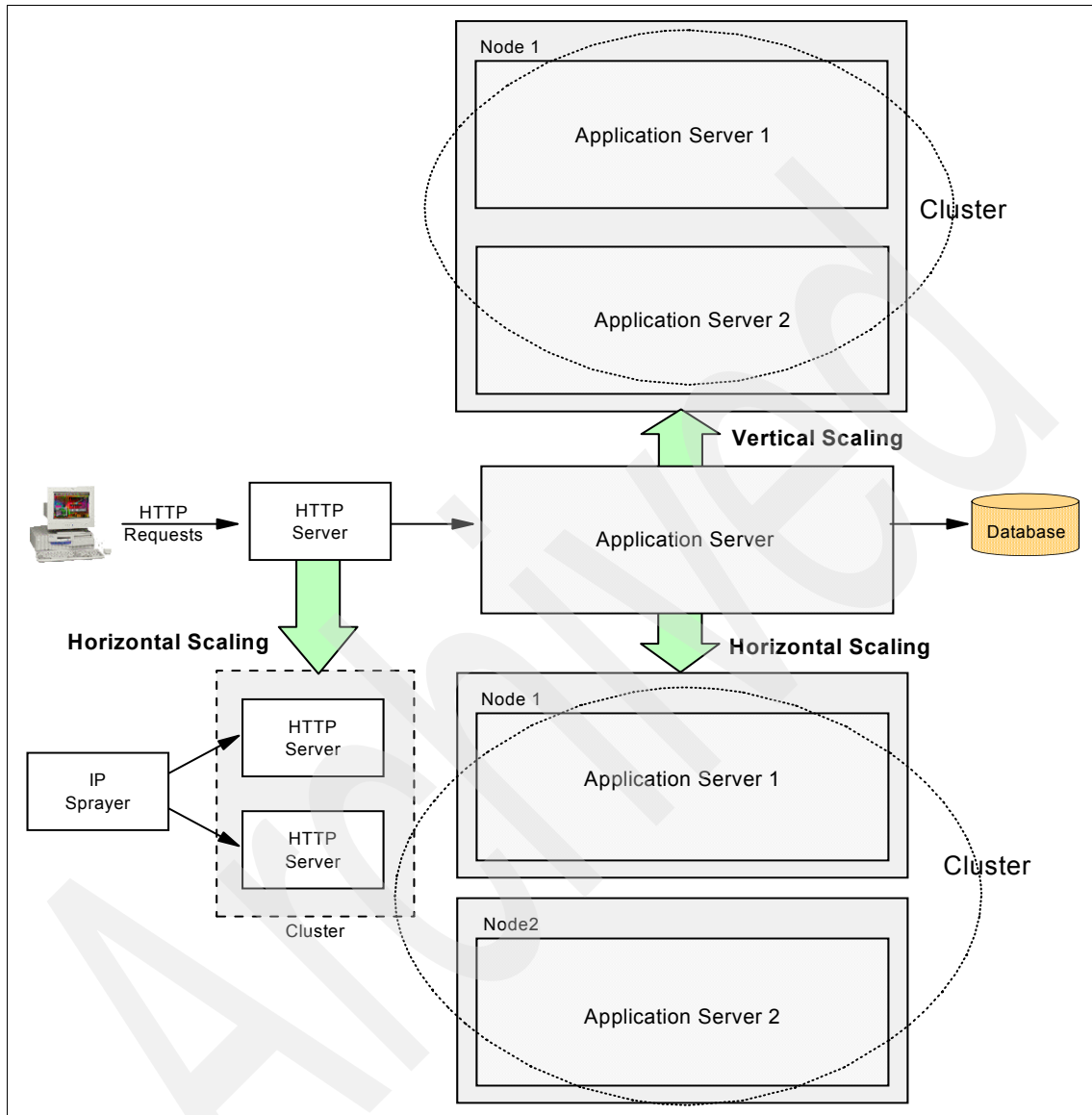


Figure 3-3 Vertical and horizontal scaling

Starting with 3.5, “Single machine, single node, Web server separated” on page 81, we describe some basic topologies, beginning with a single machine topology and expanding it to a complex topology where different techniques are applied on a set of distributed machines, in order to provide a reliable and efficient processing environment.

Session persistence considerations

If the application maintains state between HTTP requests and we are using vertical or horizontal scaling, then we must consider using an appropriate strategy for session management.

Each application server runs in its own JVM process. To allow a failover from one application server to another without logging out users, we need to share the session data between multiple processes. There are two ways of doing this in WebSphere Application Server Network Deployment:

- ▶ Memory-to-memory session replication
This method employs Data Replication Service (DRS) to provide replication of session data between the process memory of different application server JVMs. DRS is included with WebSphere Application Server and is automatically started when the JVM of a clustered (and properly configured) application server starts.
- ▶ Database persistence
Session data is stored in a database shared by all application servers.

Memory-to-memory replication has the following advantages and disadvantages compared to database persistence:

- ▶ No separate database product is required.
- ▶ Enabling and configuring replication is very easy using the Administrative Console.
- ▶ The overhead of replicating session information might be significant depending on the number of application servers in your replication domain, the size of the sessions, and the configuration of the replication domain (Single replica, Entire Domain, Number of replicas). Therefore, care must be taken to configure the replication domain correctly and database persistence might be the better choice in a memory constraint environment.

Refer to Chapter 12, “Session management”, especially Section 12.9, “Persistent session management”, of the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451 for detailed information.

3.4 Web server topology in a Network Deployment cell

A new feature in WebSphere Application Server V6 is that a Web server can be defined in a cell as a Web Server Node. This allows the association of applications to one or more Web servers. This way, custom plug-in configuration files can be generated for specific Web servers. The subsequent sections will cover some more details for this topology.

3.4.1 Web server managed node

A managed node means that the Web server is managed by the Deployment Manager. This configuration provides the ability to start and stop the Web server from the Administrative Console and automatically push the plug-in configuration file to the Web server. It requires a Node Agent to be installed on the Web server machine as shown in Figure 3-4.

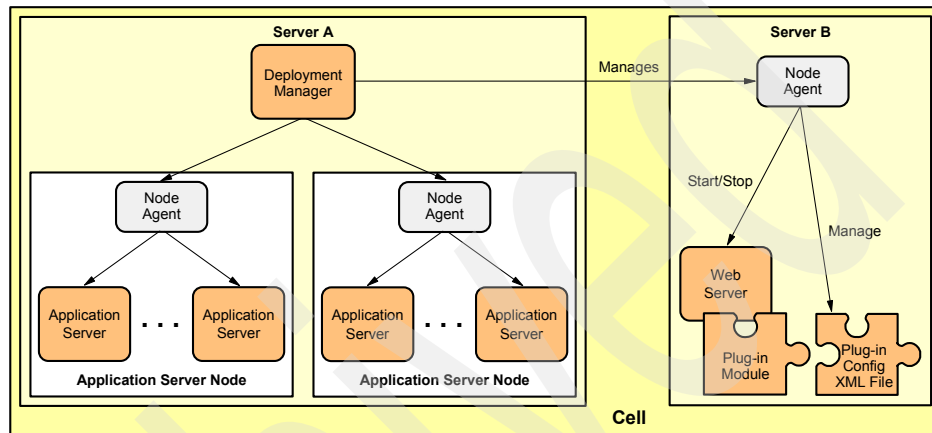


Figure 3-4 Web server managed node

3.4.2 Web server unmanaged node

This is the standard deployment option employed prior to the introduction of a managed node WebSphere Application Server V6. This is most often used for Web servers deployed between two firewalls inside a "DMZ" where no Node Agent is installed. The use of this topology requires that each time the plug-in configuration file is regenerated, it has to be copied manually from the machine where WebSphere Application Server is installed to the machine where the Web server is running. Refer to Figure 3-5 on page 80 for this topology.

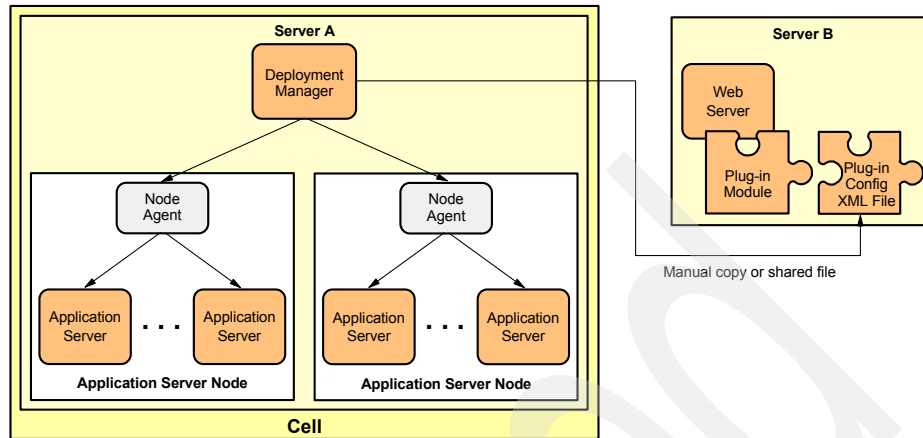


Figure 3-5 Web server unmanaged node

3.4.3 IBM HTTP Server (IHS) as unmanaged node (special case)

If the Web server is IBM HTTP Server V6, then the Web server can be installed on a remote machine without a Node Agent and still be administered from the Deployment Manager using the *IBM HTTP Server Admin Process* for tasks such as starting and stopping the Web server or automatically pushing the plug-in configuration file to it. This is shown in Figure 3-6.

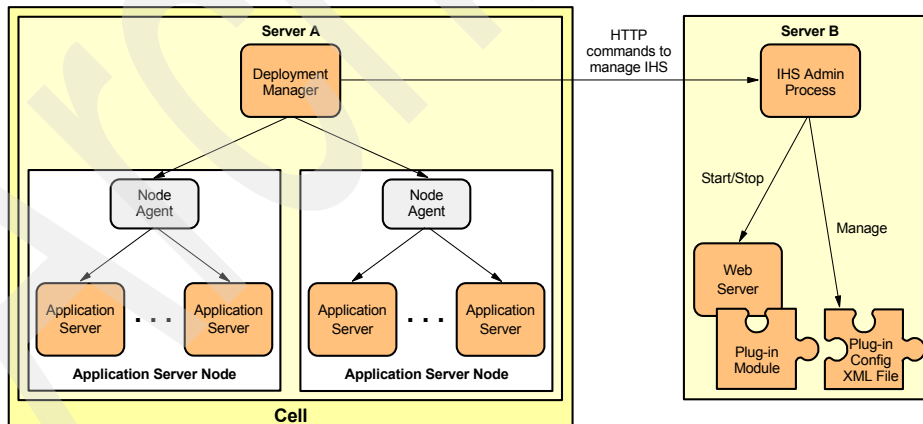


Figure 3-6 IHS unmanaged - special case

3.5 Single machine, single node, Web server separated

Although it is possible to collocate the Web server with the application server, this is not recommended, mainly for security reasons (no DMZ established). When compared to such a collocated configuration, separation of the application server and the Web server provides improvement in security, performance, throughput, availability and maintainability. Figure 3-7, “Web server separation” on page 81 illustrates this topology.

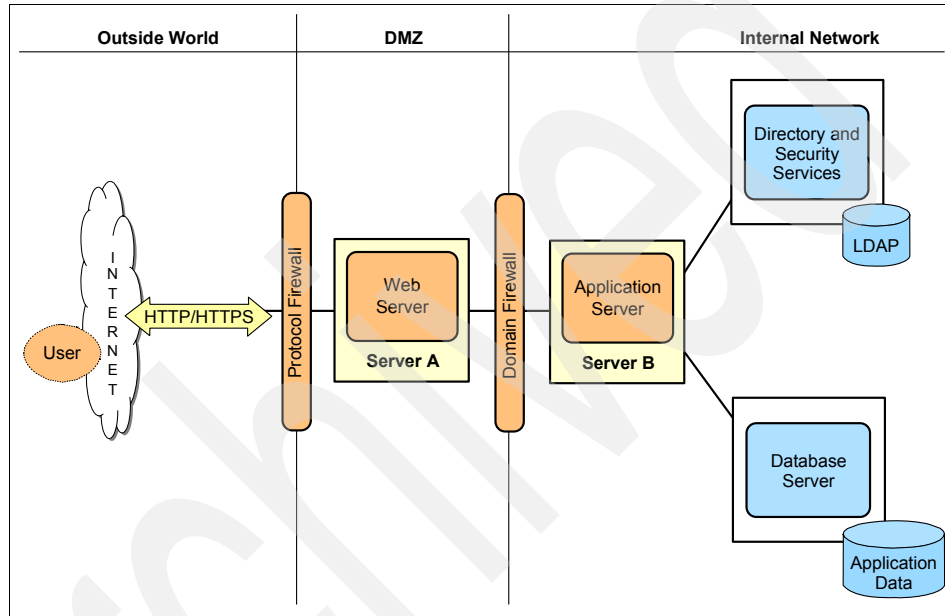


Figure 3-7 Web server separation

The Web server plug-in allows the Web server to route requests to the application server even when they are physically separated. It uses an XML configuration file (plugin-cfg.xml) that contains settings that describe how to handle and pass on requests to the WebSphere Application Server(s). Be aware that in this case, the plugin-cfg.xml configuration file is generated on the machine where the application server is installed so it has to be moved, each time it is regenerated, from the machine where the application server resides to the machine where the Web server and the plug-in module are installed.

A failure on the Web server could be bypassed pointing the DNS to the machine where WebSphere Application Server is installed. This way, the embedded WebSphere Application Server Web server (WebContainer Inbound Chain) can replace (with limited throughput) the Web server while the problem is being solved.

3.6 Vertical scaling topology

Vertical scaling refers to configuring multiple application servers on a single machine and creating a cluster of associated application servers all hosting the same J2EE application(s). This configuration is depicted in Figure 3-8.

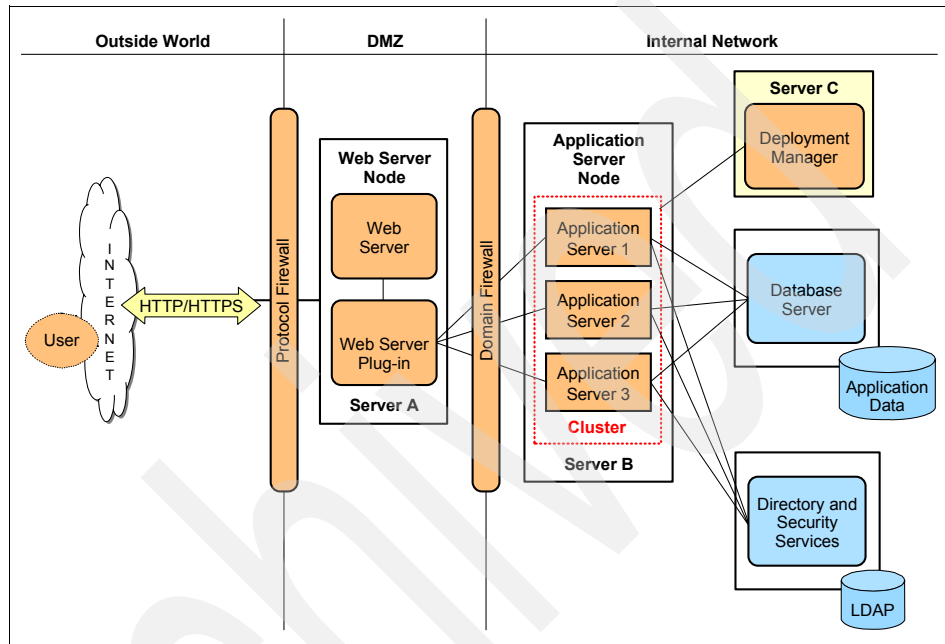


Figure 3-8 Vertical scaling

Represented in Figure 3-8 is a vertical scaling example that includes a cluster with three cluster members. In this case, the Web server plug-in routes the requests according to the application servers availability. Load balancing is performed at the Web server plug-in level based on a round-robin algorithm and with consideration of session state. Failover is also possible as long as there are active application servers (JVMs) on the system.

Vertical scaling can be combined with other topologies to boost performance, throughput and availability.

3.7 Horizontal scaling topology

Horizontal scaling exists when the cluster members are located across multiple machines. This lets a single application span over several machines, yet still presenting the application as a single logical image. Figure 3-9 on page 83

illustrates a horizontal scaling topology with two Application Server Nodes, each one on a separate machine (Servers B and C). Notice that there is a fourth server, Server D, where the Deployment Manager is installed to manage the cluster.

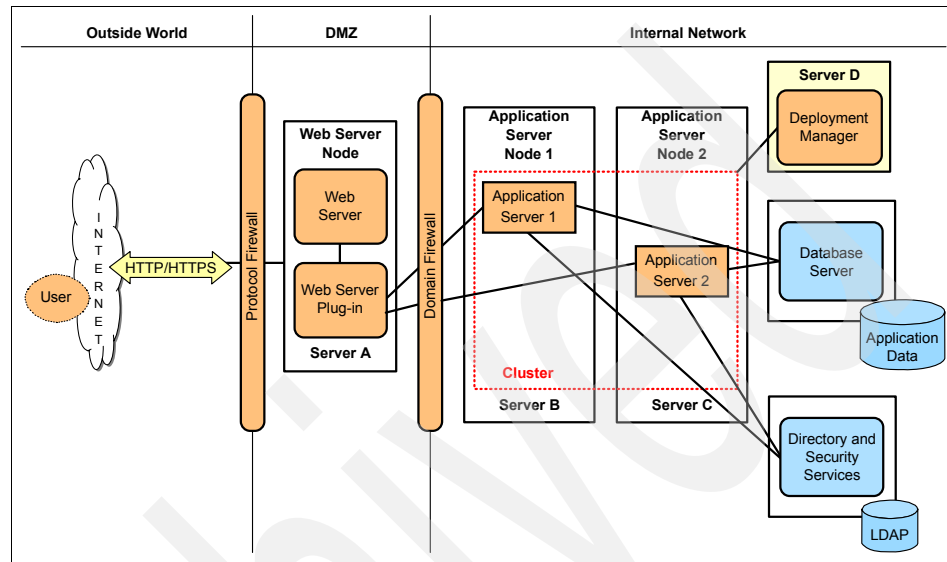


Figure 3-9 Horizontal scaling with cluster

The Web server plug-in distributes requests to the cluster members on each node and performs load balancing and failover. If the Web server (Server A) goes down, then the WebContainer Inbound Chain of Server B or C could be utilized (limited throughput) meanwhile Server A or the Web server on Server A is repaired.

Be aware that this configuration introduces a single point of failure; when the HTTP server is out of service, your entire application is inaccessible from the outside network (internal users could still access the application server(s) using the WebContainer Inbound Chain). You can omit this SPOF by adding a backup Web server.

If any component in the Application Server Node 1 (hardware or software) fails, the Application Server Node 2 can still serve requests from the Web Server Node and vice versa.

The Load Balancer, part of the WebSphere Edge Components, can be configured to create a cluster of Web servers and add it to a cluster of application servers. This is shown in 3.8, “Horizontal scaling with IP sprayer topology” on page 84.

3.8 Horizontal scaling with IP sprayer topology

Load balancing products can be used to distribute HTTP requests among Web servers that are running on multiple physical machines.

The Dispatcher component of Load Balancer, which is part of the WebSphere Edge Components, is an IP sprayer that performs intelligent load balancing among Web servers based on server availability and workload capacity as the main selection criteria to distribute the requests. Refer to Chapter 4, “Introduction to WebSphere Edge Components” on page 99 for details.

Figure 3-10 illustrates a horizontal scaling configuration that uses an IP sprayer on the Load Balancer Node to distribute requests between Web servers on multiple machines.

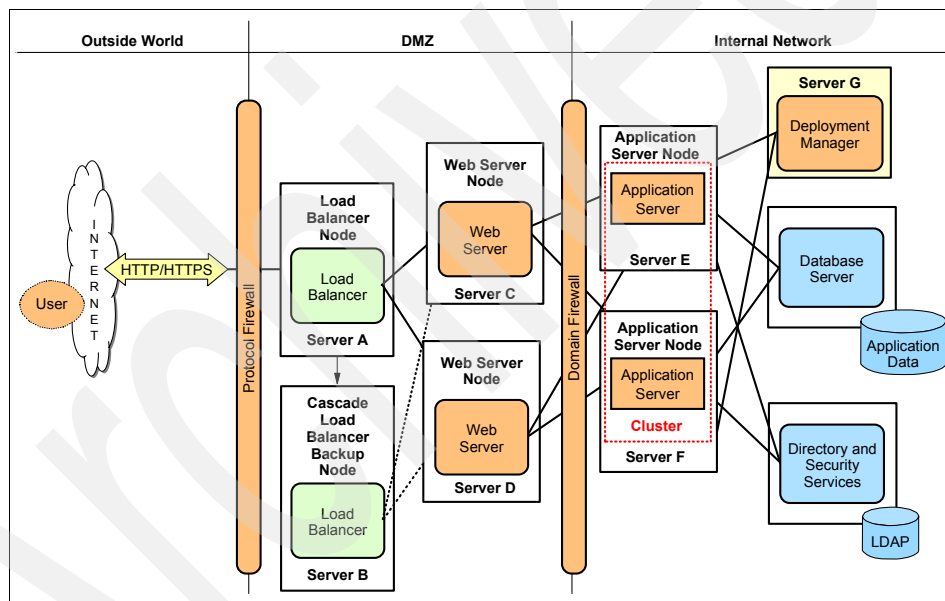


Figure 3-10 IP sprayer horizontally scaled topology

The Load Balancer Node sprays Web client requests to the Web servers. The Load Balancer is configured in cascade. The primary Load Balancer communicates to his backup through a heartbeat to perform failover, if needed, and thus eliminates the Load Balancer Node as a single point of failure.

Both Web servers perform load balancing and failover between the application servers (cluster members) through the Web server plug-in.

Tip: The Web server and Load Balancer can be collocated.

If any component on Server C, D, E, or F fails, the other ones can still continue receiving requests.

3.9 Topology with redundancy of several components

The idea behind having as much redundancy of components as possible is to eliminate (keep minimized) the single points of failure (SPOF). Most of the components allow some kind of redundancy like a Load Balancer backup node for the primary Load Balancer node, or clustered Web servers and/or application servers, etc. Some other components like the Deployment Manager do not support any automatic backup/failover. Figure 3-11 on page 86 illustrates a topology with redundancy of several components including:

- ▶ **Two Load Balancers**
The one on Server A is the primary (active) Load Balancer. It is synchronized, through a heartbeat, with a backup Load Balancer (in standby status) on another machine, Server B.
- ▶ **Two Web servers**
Both of them receive requests from the Load Balancer and share the requests that come from the Internet. Each one is installed on a different machine.
- ▶ **An application server cluster**
The cluster implements vertical and horizontal scaling.
- ▶ **Eight cluster members**
Two on each Application Server Node.
- ▶ **Four Application Server Nodes**
Each one hosting two application servers. The nodes on Server E are independent installations. The nodes on Server F are profiles of a single installation.
- ▶ **Two database servers**
Using a HA (high availability) software product. This means that one copy of the database is the one that is being used and the other one is a replica that will replace the first one if it fails.
- ▶ **Two LDAP servers**
Using a HA (high availability) software product. This means that one copy of the database is the one that is being used and the other one is a replica that will replace the first one if it fails.

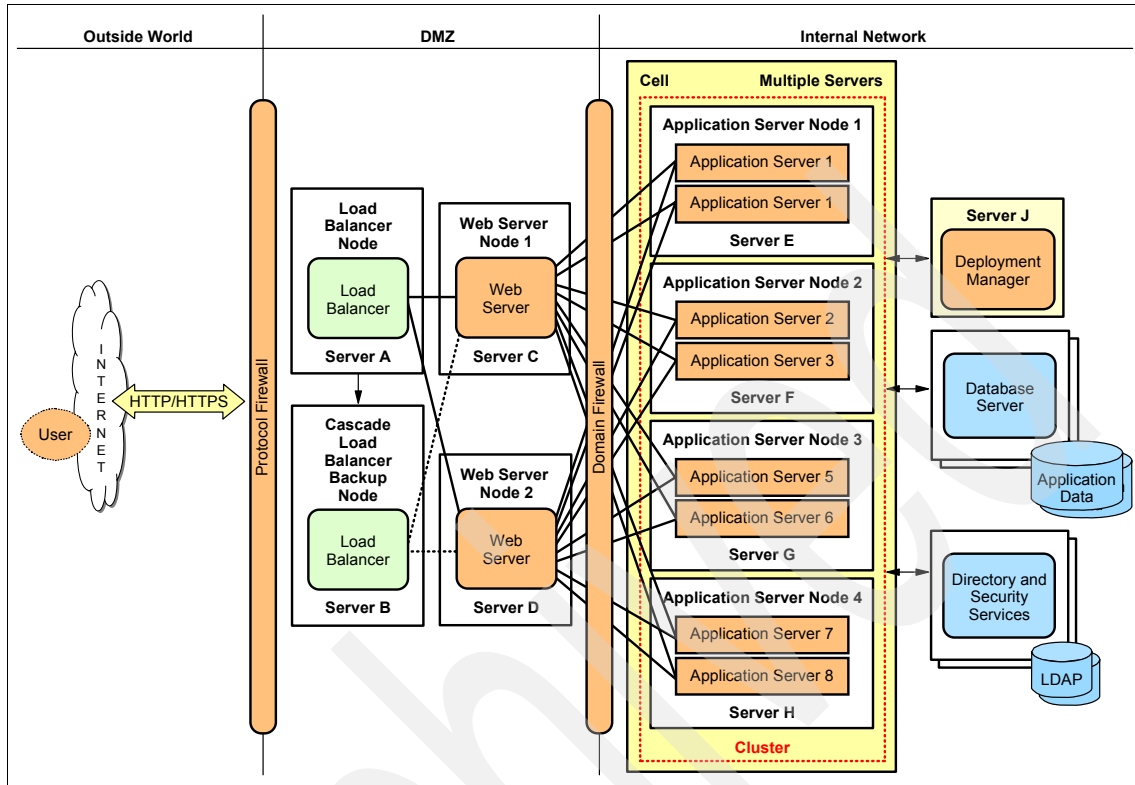


Figure 3-11 Topology with redundancy of several components

3.10 The sample topology

We started our discussion of topologies with a single server configuration where all components reside on the same machine. Now we introduce a sample configuration that explores some of the subjects discussed so far:

- ▶ Separating the HTTP server
- ▶ Separating the database
- ▶ Vertical scaling
- ▶ Horizontal scaling
- ▶ HTTP server clustering

To configure, test, and evaluate the behavior of several components, we set up the configuration illustrated in Figure 3-12 on page 88 and used it throughout this book to test and evaluate aspects of IBM WebSphere Application Server V6 scalability and high availability. Chapter 8, “Implementing the sample topology” on page 387, describes the steps to set up this sample configuration.

The resulting topology is composed of:

- ▶ A cluster of two Web servers (IBM HTTP Server) supplemented by a Caching Proxy and a Load Balancer (both from WebSphere Edge Components). Both the Caching Proxy and the Load Balancer are installed with backup servers to provide high availability of these components.
- ▶ A dedicated Deployment Manager machine managing the WebSphere Application Server cell, running IBM WebSphere Application Server Network Deployment V6.
- ▶ A WebSphere cluster of three application server processes on two physical machines. WebSphere Application Server V6 is installed on both machines.

A dedicated database server running IBM DB2 UDB V8.2.

Important: To make it easier to demonstrate the workload management and failover capabilities of WebSphere, we have chosen to split the Web container and the EJB container in our sample topology.

As mentioned earlier, while this is a possible configuration, however, it is *not* recommended because it will most likely have a negative impact on the performance. This is as a result of the out of process calls from the EJB clients in the Web container to the EJB container.

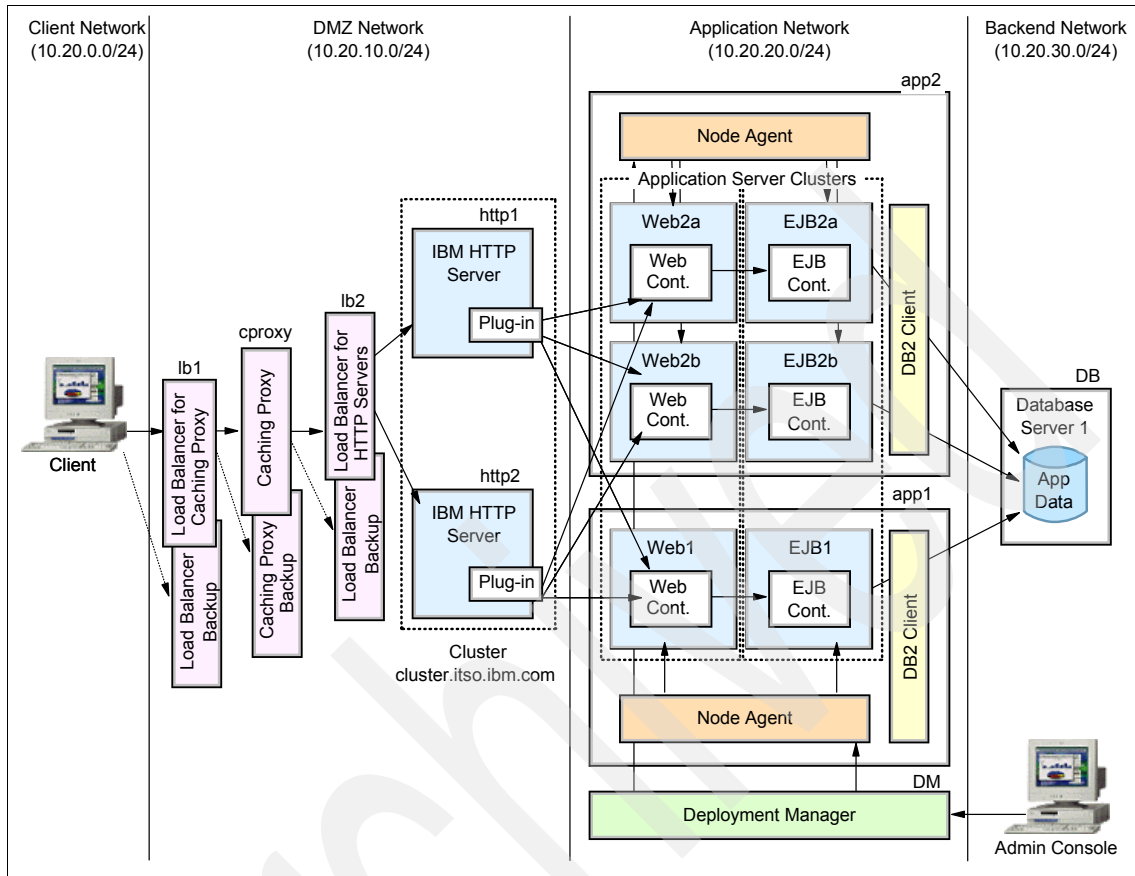


Figure 3-12 ITSO sample topology

We have two HTTP servers configured as a cluster under the control of WebSphere Load Balancer. The HTTP servers are configured to respond both to port 80 (HTTP) and port 443 (HTTPS). The Load Balancer server distributes requests among the caching proxies and HTTP servers based on weights set by the Load Balancer's Manager component.

In addition to monitoring the network reachability of the machines, an HTTP advisor is activated to poll the availability of the HTTP service on each machine.

3.11 Topologies and high availability

In order to select the most appropriate topology, it is important to understand how WebSphere provides high availability for the Web container and EJB container. Therefore, we examine the three failure situations shown in Figure 3-13: HTTP server, Web container, and EJB container.

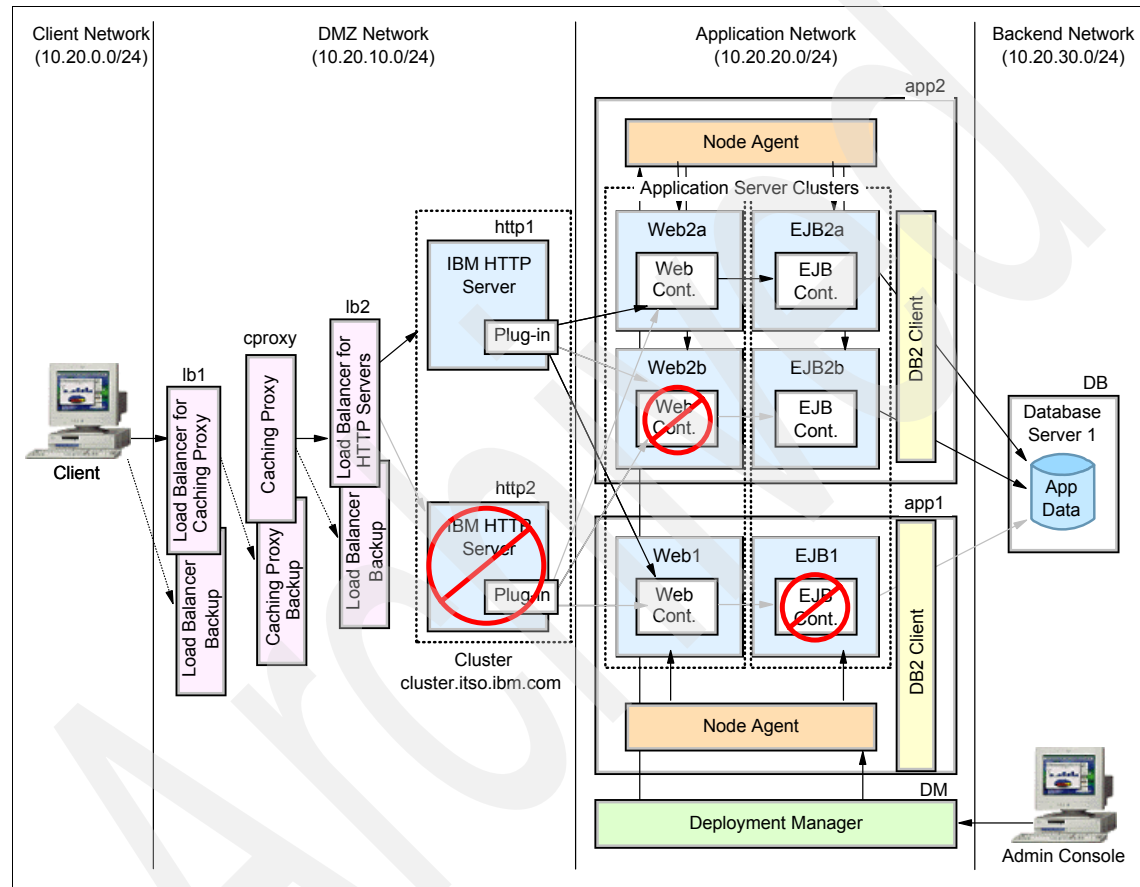


Figure 3-13 Failure situations

In normal operation, the HTTP requests are forwarded to one of the clustered machines (http1 or http2) acting as HTTP servers. If the requests should be served by WebSphere, the plug-in uses HTTP transport to route requests to the application servers on system app1 or app2. The Web applications are deployed on member application servers in the WEBcluster (Web1 on app1, and Web2a and Web2b on app2).

In the case of a failure of one of these application servers, or if the application server is not active, the plug-in redirects the requests to the available members in the same cluster (WEBcluster). The requests will be sent using HTTP, either to the same machine or to a separate machine. For plug-in workload management and failover details, refer to Chapter 6, “Plug-in workload management and failover” on page 227.

The same applies to EJB containers. We configured a cluster consisting of applications servers in EJBcluster (EJB1 on system app1, and EJB2a and EJB2b on app2). If an EJB application server is not available, requests are directed to the available EJB application server in the EJBcluster. Chapter 7, “EJB workload management” on page 341 provides further details on EJB workload management.

If an HTTP server fails, Load Balancer redirects HTTP requests to the available HTTP servers on the cluster. Figure 3-14 on page 91 shows how requests are routed around the failed Web server and application servers in our sample topology.

Note: Failure of the Deployment Manager does not interrupt the managed servers. The Deployment Manager node stores the master configuration repository and every managed server stores only the appropriate subset of the master configuration. Therefore, we do recommend maintaining the master configuration repository on a shared file server. The objective is to recover the configuration data, in exceptional cases such as a complete failure of the machine running Deployment Manager. See the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451 for more information about the configuration repository.

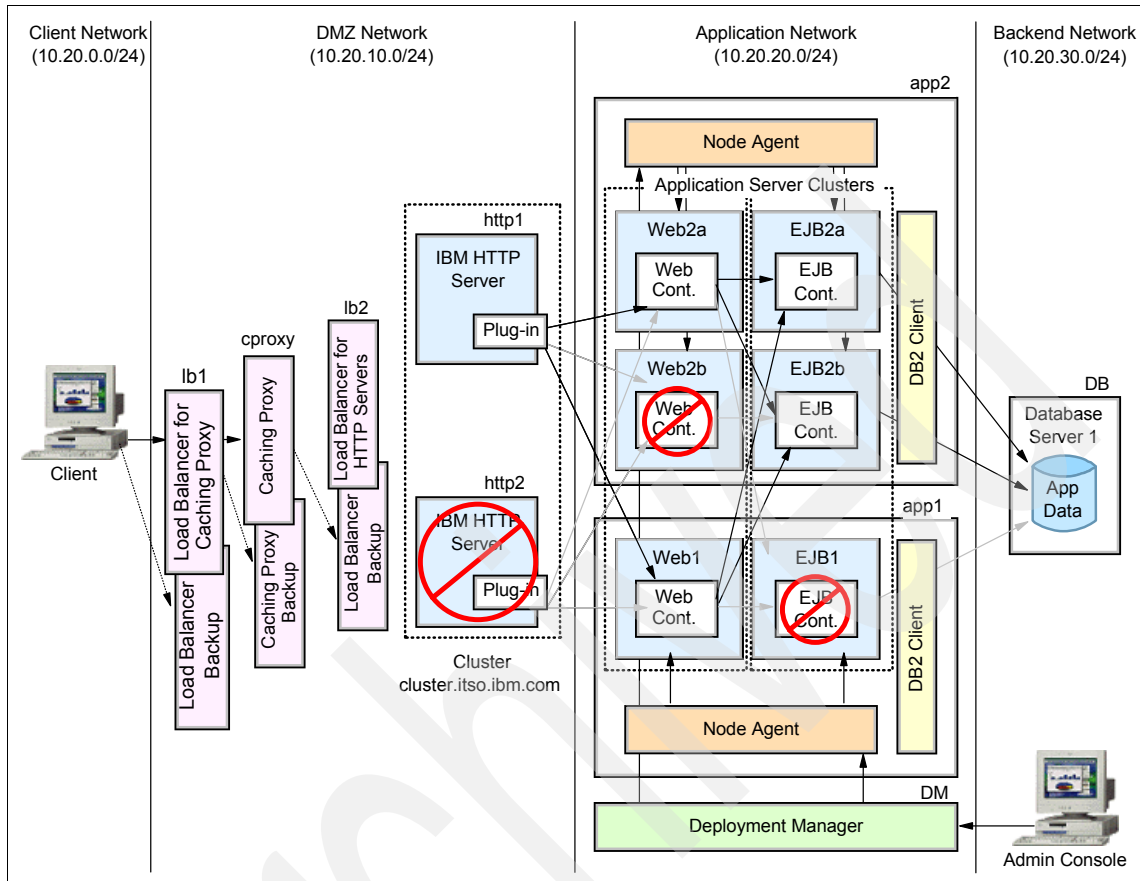


Figure 3-14 Failover behavior

3.11.1 Using WebSphere Load Balancer custom advisor

If the WebSphere machine has stopped or the application server is down while the HTTP server is still running, you will receive an error message on the browser, as illustrated in Figure 3-15 on page 92.

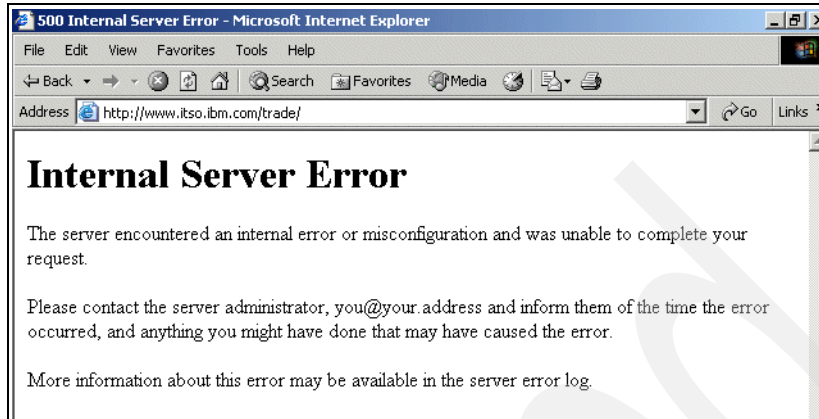


Figure 3-15 Message when application server is down

To help avoid this problem, WebSphere Load Balancer provides a sample custom advisor for WebSphere Application Server. We used this advisor instead of the default HTTP advisor. The basic layout (without back-up servers) of our solution is shown in Figure 3-16.

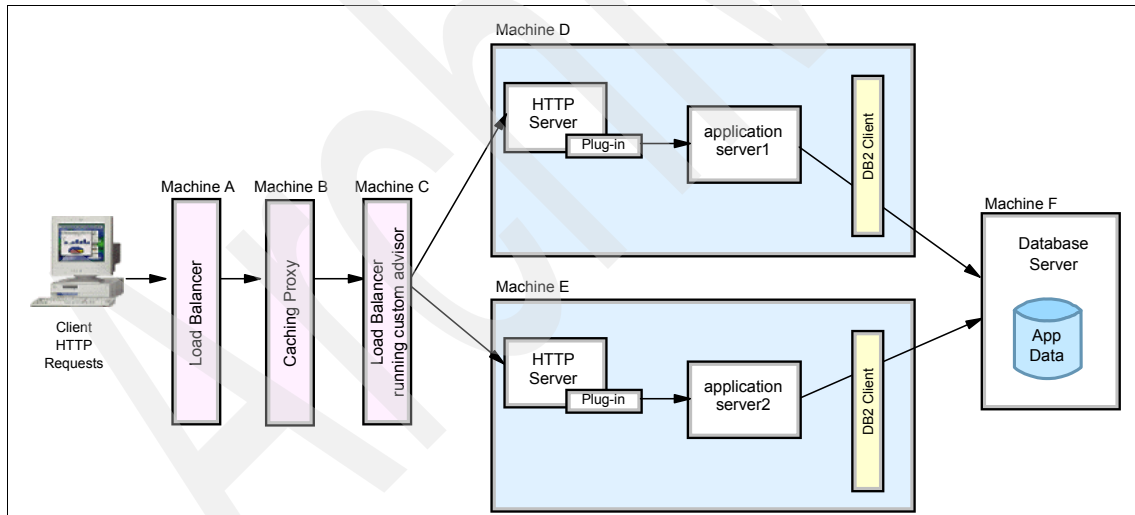


Figure 3-16 Using WebSphere custom advisor

Now, when the WebSphere custom advisor is running, we can continue to access the application server2 on machine E without getting the error message even when machine D is stopped. The custom advisor directs the request to the HTTP server on machine E, as shown in Figure 3-17 on page 93.

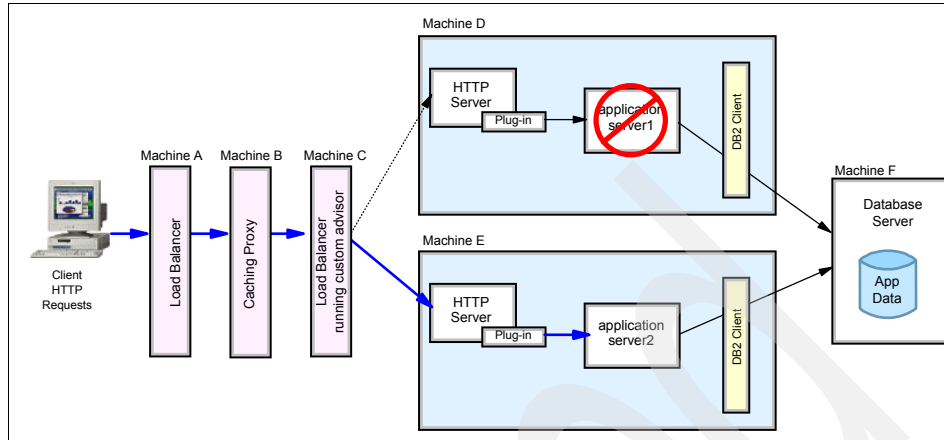


Figure 3-17 Requests dispatched to active application server

Notes:

- ▶ You cannot run the HTTP advisor and the WebSphere custom advisor at the same time if you specify the same port number for both advisors.
- ▶ WebSphere custom advisor must be considered as a monitoring extension associated with each HTTP server on the cluster. The custom advisor prevents requests from being sent to a specific HTTP server when this HTTP server cannot appropriately fulfill them (for example, when the WebSphere server it sends requests to is down).

When we use WebSphere workload management, the requests from multiple HTTP servers are sent to a group of application servers, distributed also among multiple machines. We cannot associate the service layer provided by the application server to an HTTP server anymore, since the plug-in is responsible for distributing the requests.

WebSphere custom advisor can have a more meaningful use when WebSphere workload management is not used or a whole cluster associated with a specific HTTP server is down.

When using WebSphere plug-in workload management, as in our sample topology, monitoring the HTTP servers using the HTTP advisor is probably the best choice.

For more information about this topic, please refer to Chapter 5, “Using IBM WebSphere Edge Components” on page 127.

3.12 Topology selection summary

Table 3-1 to Table 3-5 on page 96 provide a summary of possible considerations for topology selection as discussed in this chapter. These tables list the requirements (such as availability, performance, security, maintainability, and session persistence) and the possible solution(s) for the Web server, the application server(s) and the database server:

Table 3-1 Topology selection based on availability requirements

Requirement = Availability	Solution/Topology	Where do you find more information?
Web server	Either Edge Components Load Balancer (with backup) or high availability (HA) solution, based on your other requirements	<ul style="list-style-type: none">▶ Chapters 4 and 5 (Edge Components)▶ <i>WebSphere Application Server Network Deployment V6: High availability solutions</i>, SG24-6688
Application server	<ul style="list-style-type: none">▶ Vertical scaling (process redundancy)▶ Horizontal scaling (hardware and process redundancy)▶ A combination of both	<ul style="list-style-type: none">▶ Chapters 8 and 9 of the redbook <i>IBM WebSphere V6 Planning and Design Handbook</i>, SG24-6446▶ Chapters 6, 7, and 8 (Workload management and Sample Topology)
Database server	HA software solution	<ul style="list-style-type: none">▶ <i>WebSphere Application Server Network Deployment V6: High availability solutions</i>, SG24-6688

Table 3-2 Topology selection based on performance requirements

Requirement = Performance and Throughput	Solution/Topology	Where do you find more information?
Web server	<ul style="list-style-type: none">▶ Multiple Web servers in conjunction with Edge Components Load Balancer▶ Caching proxy▶ Dynamic caching with AFPA or ESI external caching	<ul style="list-style-type: none">▶ Chapters 4 and 5 (Edge Components)▶ Chapter 10 (Dynamic caching)

Requirement = Performance and Throughput	Solution/Topology	Where do you find more information?
Application server	<ul style="list-style-type: none"> ▶ Clustering (in most cases horizontal) ▶ Dynamic caching ▶ Serving static content from Web server to offload application server 	<ul style="list-style-type: none"> ▶ Chapters 8 and 9 of the redbook <i>IBM WebSphere V6 Planning and Design Handbook</i>, SG24-6446 ▶ Chapters 6, 7, and 8 (Workload management and Sample Topology) ▶ Chapter10 (Dynamic caching) ▶ Technote TIPS0223 (Separating Static and Dynamic Web Content) at http://publib-b.boulder.ibm.com/Redbooks.nsf/RedbookAbstracts/tips0223.html?open
Database server	Separate DB server	<ul style="list-style-type: none"> ▶ Chapters 8 and 9 of the redbook <i>IBM WebSphere V6 Planning and Design Handbook</i>, SG24-6446

Table 3-3 Topology selection based on security requirements

Requirement = Security	Solution/Topology	Where do you find more information?
Web server	Separate the Web server into the DMZ - either on LPAR or separate system	<ul style="list-style-type: none"> ▶ Chapters 8 and 9 of the redbook <i>IBM WebSphere V6 Planning and Design Handbook</i>, SG24-6446
Application server	not covered	<ul style="list-style-type: none"> ▶ Redbook <i>WebSphere Application Server V6: Security Handbook</i>, SG24-6316
Database server		

Table 3-4 Topology selection based on maintainability requirements

Requirement = Maintainability	Solution/Topology	Where do you find more information?
Web server	Single machine environment is easiest to maintain, but can only be combined with horizontal scaling	► Chapters 8 and 9 of the redbook <i>IBM WebSphere V6 Planning and Design Handbook</i> , SG24-6446
Application server		
Database server		

Table 3-5 Topology selection based on session affinity requirements

Requirement = Session persistence	Solution/Topology	Where do you find more information?
Web server	Session affinity is handled by the WebSphere plug-in	► Chapter 6 (Plug-in workload management)
Application server	<ul style="list-style-type: none"> ► Session database (possible SPOF if not HA) ► Memory-to-memory replication 	<ul style="list-style-type: none"> ► Chapters 6 (Plug-in workload management) ► Chapter 12 of the redbook <i>WebSphere Application Server V6 System Management and Configuration Handbook</i>, SG24-6451
Database server	n/a	► n/a

As mentioned earlier, for detailed information about topology selection criteria, please refer to the redbook *IBM WebSphere V6 Planning and Design Handbook*, SG24-6446.



Part 2

Distributing the workload

ARCHIVED

Introduction to WebSphere Edge Components

In this chapter, we describe the functions and possibilities offered by the IBM WebSphere Edge Components which are part of IBM WebSphere Application Server Network Deployment V6. We provide an introduction to the following functions of Edge Components:

- ▶ Load Balancer
- ▶ Caching Proxy

Details on how to configure and use these components are covered in Chapter 5, “Using IBM WebSphere Edge Components” on page 127.

4.1 Introduction

In a real-world environment, when the number of users accessing your Web site increases, you experience slow response times. Your site may even fail under critical load conditions. Simply increasing the processing power and other resources on your server may no longer be cost effective. You need to provide scalability for your environment and ensure that it has the required availability and provides better performance.

4.1.1 Scalability

Applications need to scale for increasing numbers of simultaneous users on a wide range of Web access devices.

By adding one or more Web servers to the existing environment, you can prevent a single Web server from becoming overloaded. The incoming requests are then dispatched to a group of servers, called a *cluster*. A cluster is a group of independent nodes interconnected and working together as a single system.

Load balancer software is used to dispatch the load to the Web servers in the cluster. It uses a load balancing mechanism usually known as *IP spraying*, which intercepts the HTTP requests and redirects them to the appropriate machine in the cluster, providing scalability, load balancing, and failover.

4.1.2 Availability

Users must be able to reach the application regardless of failed servers. In a clustered Web server environment, the load balancer monitors the availability of the Web servers. If a Web server has failed, no more requests are sent to it. Instead, all requests are routed to the remaining active Web servers. It is also recommended that you ensure high availability of the load balancer system itself to eliminate it as a single point of failure (SPOF).

4.1.3 Performance

Quick response times can be provided by routing requests based on the geographic location, user identity, or content requested and by caching the retrieved data.

4.2 IBM WebSphere Edge Components overview

WebSphere Edge Components, which are part of IBM WebSphere Application Server Network Deployment V6, help you to reduce Web server congestion, increase content availability, and improve Web server performance.

The following products are included in the Edge Components of IBM WebSphere Application Server Network Deployment V6:

- ▶ Load Balancer
- ▶ Caching Proxy

WebSphere Edge Components are supported on a wide range of operating systems, such as AIX, HP-UX, Linux (on various platforms, such as Intel, zSeries, iSeries, and pSeries), Solaris, Windows 2000 and Windows 2003. For more information about the supported platforms and product requirements, refer to *Concepts, Planning, and Installation for Edge Components Version 6.0*, GC31-6855.

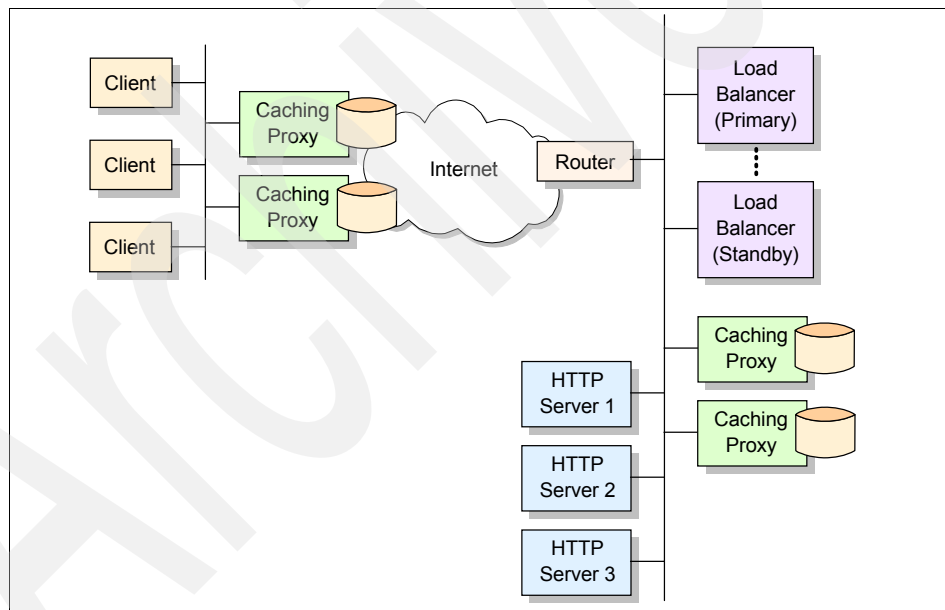


Figure 4-1 Edge Components

Details about the Load Balancer can be found in 4.3, “Load Balancer overview” on page 102, and about the Caching Proxy in 4.8, “Caching Proxy overview” on page 122. Both components are covered in more detail in Chapter 5, “Using IBM WebSphere Edge Components” on page 127.

4.3 Load Balancer overview

Load Balancer consists of the following five components that can be used separately or together:

- ▶ Dispatcher (see 4.3.1, “Dispatcher” on page 102).
- ▶ Content Based Routing (CBR) for HTTP and HTTPS (see 4.3.2, “Content Based Routing (CBR)” on page 109).
- ▶ Site Selector (see 4.3.3, “Site Selector” on page 110).
- ▶ Cisco CSS Controller (see 4.3.4, “Cisco CSS Controller and Nortel Alteon Controller” on page 110).
- ▶ Nortel Alteon Controller (see 4.3.4, “Cisco CSS Controller and Nortel Alteon Controller” on page 110).

We cover these components in more detail in the following sections.

Session affinity is an option that applies to all of these components. See 4.4, “Server affinity in Load Balancer” on page 110 for details.

4.3.1 Dispatcher

The Dispatcher component distributes the load it receives to servers contained in a cluster (a set of servers that run the same application and can provide the same contents to its clients). This mechanism is also known as IP spraying.

Note: Load balancing can handle any TCP/IP compliant protocol, not only HTTP and HTTPS. For example, Dispatcher can provide load balancing for such protocols as FTP, NNTP, IMAP, POP3, SMTP, Telnet, and so on.

Dispatcher decides which server will handle a certain TCP/IP connection based on the weight of each server in the cluster. The weight is the value that determines the number of connections that each server receives. The weight can be fixed in the configuration or it can be dynamically calculated by Dispatcher.

If you choose to configure the weight of the servers and set it as a fixed value, it will not change no matter the conditions of the balanced servers. For example, if you configure a cluster containing two servers, and you set the weight of the first server to 1, and the weight of the second server to 2, meaning that the second server will always receive twice the load as the first server. The only exception to this is when an Advisor detects a failed server.

If you choose to work with dynamic weights (which is the default option), Dispatcher will calculate the load of each balanced server dynamically. In our previous example, if the response time of the second server was slower than the

response time of the first server, it would now be possible to detect this and generate the correct weight value according to the real conditions of each server.

Dispatcher's internal components

Dispatcher has internal components that are responsible for the tasks mentioned earlier, like distributing TCP/IP packets and calculating the weight of the balanced servers. These components are:

- ▶ Executor
- ▶ Manager
- ▶ Advisors
- ▶ Metric Server

Executor

Executor is the core component of Dispatcher, and it is responsible for the load distribution. It receives the packet, identifies if this packet is destined to the operating system or if it is destined to a cluster. If the packet is destined to a cluster, it will then determine whether this packet is a follow up to an existing connection, or if it is a request for a new connection. Executor keeps a connection table in memory to keep track of all active connections. After that, it chooses the back-end server to which this packet will be sent.

In order to be able to identify the packets meant for the operating system, the administrator needs to associate an IP address to the variable *NFA* (non-forwarding address). This variable contains the IP address that is used for all connections that should not be load balanced by Dispatcher, like telneting into the machine, connecting to the Dispatcher's administration service, etc. In other words, NFA determines the IP address that the Executor will ignore as far as load balancing is concerned.

Manager

Manager is the component responsible for providing weight values of each balanced server to Executor, so it can make its load balancing decision. Running this component is optional, but it is necessary for dynamic weighting of the servers and also for identifying failed servers.

Manager uses four metric values for calculating the weight value of each server: the number of active connections being handled by that server, the number of new connections that were forwarded to that server since the last check (the default is two seconds) and the input from two components that gather load information about the balanced servers: the Advisors and the Metric Server.

Advisors

The Advisors are lightweight clients that run on the Dispatcher server, and they are aware of the protocol used by the back-end servers. Load Balancer provides advisors for HTTP, HTTPS, FTP, LDAP, among others.

Each advisor connects to a certain service running on each server of the cluster, and submits a request that will validate the health of that service. This means that the advisor actually tests the service, not only the connectivity to the server (a system can be reachable by ping, but if the Web server is not running, it cannot be used in load balancing). The advisor then returns a value to the Manager, which represents how long it took for each server to respond. If it does not receive a response from a server, it will provide a value of -1 for this server, which is interpreted by the Manager as a server being down. Refer to “Advisors” on page 107 for more information about the Advisors and to 5.2, “Load Balancer configuration: basic scenario” on page 135 for usage examples.

Metric Server

If you need to collect more information from the back-end server for the load balancing, you can also use the Metric Server, which is a component that is installed and runs in each back-end server. Metric Server can additionally provide values for the server where it is running. For example, Metric Server can monitor memory and CPU usage. This information is also sent to the Manager and it is used to calculate the final weight value for each server.

The interaction of Executor, Manager, and other Load Balancer components is shown in Figure 4-2.

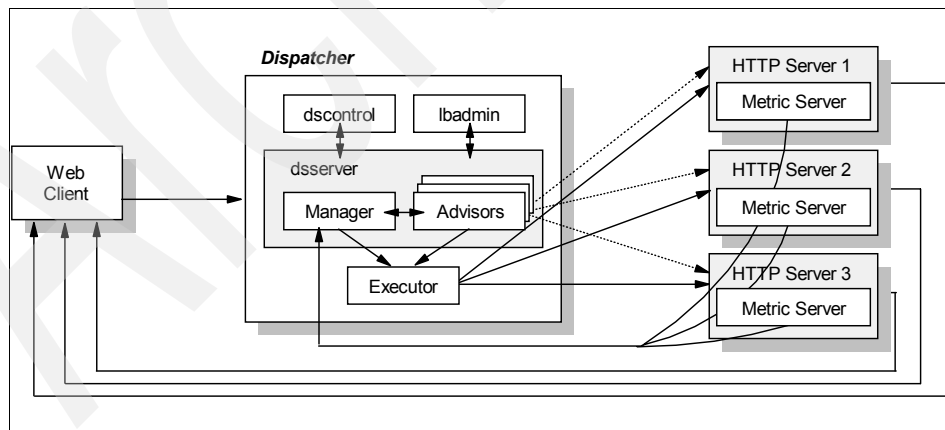


Figure 4-2 Dispatcher components interaction

Forwarding methods

There are three methods used by Executor to forward packets to the balanced servers:

- ▶ MAC forwarding
- ▶ Network Address Translation (NAT)/ Network Address Port Translation (NAPT)
- ▶ Content Based Routing (CBR), also referred to as Kernel CBR (KCBR) in previous versions

MAC forwarding

This is the default forwarding method. When Dispatcher receives a packet and chooses which server to send it to, it only changes the source and destination MAC address of the packet; the IP addresses remain the same. This means that the source IP address remains the IP address of the client machine, and the destination IP address remains the cluster IP address.

When the balanced server receives the packet, it responds directly to the client (because the source IP address in the packet belongs to the client).

MAC forwarding is the fastest forwarding method because Dispatcher receives only the incoming traffic. All outbound traffic is sent directly from the balanced server to the client. This requires that all balanced servers be connected to the same subnet as Dispatcher.

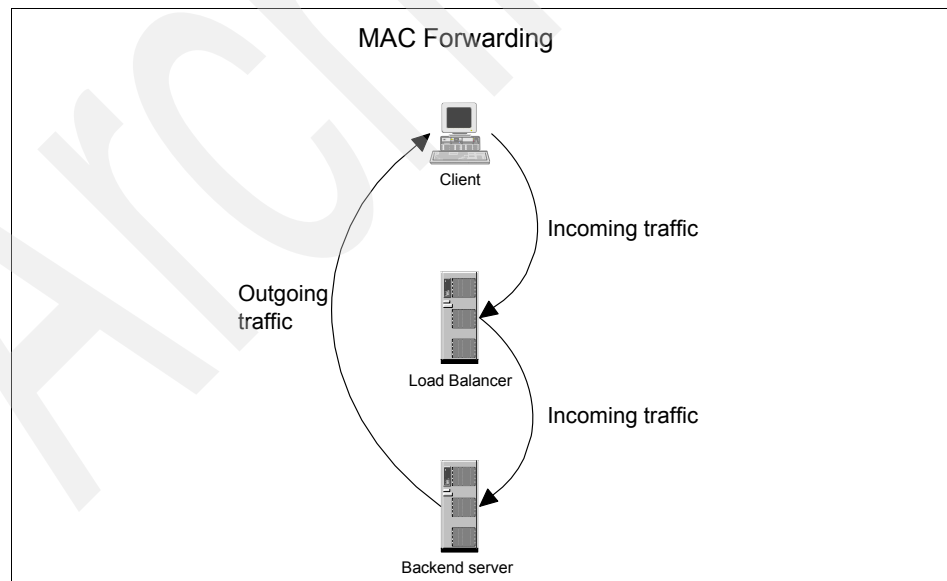


Figure 4-3 MAC forwarding - Network flow

This method also requires that the services running on the balanced servers be able to accept the packets containing the cluster IP address as the destination IP address. The easier solution is to add an IP alias to the loopback interface (so it is not advertised in the network). Refer to 5.2.2, “Configuring the balanced servers” on page 148 or *Load Balancer Administration Guide Version 6.0*, GC31-6858 for instructions on how to add an IP alias in various operating systems.

Network Address Translation (NAT)/ Network Address Port Translation (NAPT)

This forwarding method allows Dispatcher to provide load balancing for remote servers, which is not available in the MAC forwarding method.

Dispatcher receives the TCP/IP packet and chooses which server to send it to, then it rewrites the IP header and changes the source IP address (which is originally the IP address of the client machine), puts the return address instead (this is an IP address configured by the Dispatcher administrator), changes the destination IP address (which is originally the IP address of the cluster) and puts the balanced server IP address instead. Now this packet can be routed to the balanced server even if it is on a remote network. But because Dispatcher changes the packet, it needs to receive the response so it can also change the IP header before sending it to the client.

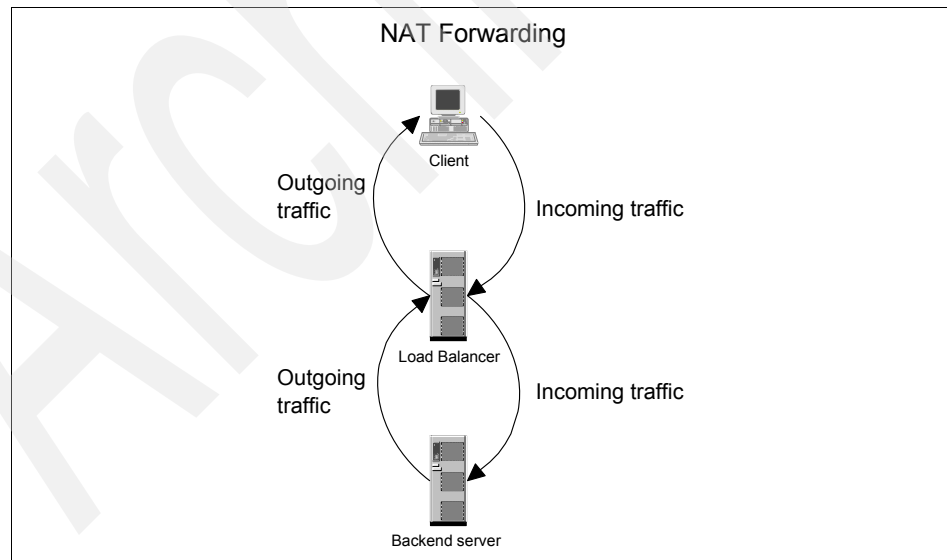


Figure 4-4 NAT forwarding - Network flow

This method also allows port redirection (NAPT). This means that the port that you configure on the cluster configuration does not need to be the same port that the service is listening on in the balanced server. In this case, Dispatcher changes the port information in the TCP header the same way it does with the IP addresses in the IP header of the TCP/IP packet.

This method implies that Dispatcher needs to handle all traffic, both inbound and outbound. It also needs one extra IP address to implement the configuration, which is the return address.

Content Based Routing (CBR)

The CBR forwarding method does not require Caching Proxy, as does the CBR component (refer to 4.3.2, “Content Based Routing (CBR)” on page 109). It allows content-based load balancing for HTTP and HTTPS protocols.

For the HTTP protocol, the connection distribution is based on the contents of the URL or the HTTP header. For the HTTPS protocol, the distribution is based on the SSL session ID field of the client request.

CBR also allows load distribution to servers connected to remote networks; it also requires one IP address for the return address.

Note: By default, the only available forwarding method is MAC forwarding. In order to enable NAT/NAPT and CBR, you need to configure the client gateway property of Executor, and set it to the IP address of the router of the network.

Refer to 5.4, “Load Balancer: NAT scenario” on page 181 for more details on how to enable all available forwarding methods.

For more details, advantages and disadvantages of each forwarding method, refer to the *Load Balancer Administration Guide Version 6.0*, GC31-6858.

Advisors

Advisors are lightweight clients that run on the Dispatcher machine, providing information about the load of a given server. The product provides protocol-specific advisors for several protocols and products, such as HTTP, HTTPS, FTP, Telnet, DB2, DNS, LDAP, SMTP, and others.

Standard advisors send transactions periodically to determine the status of the servers (for example, for HTTP an HTTP HEAD request is sent, and for FTP a SYST command is sent). If the transaction succeeds, the server is considered up.

Load Balancer also provides a generic advisor, called Connect, that can be used in case you need to load balance a service or protocol for which there is no dedicated advisor available. Connect opens a connection to the server using the server port informed in the advisor configuration and closes the connection after the TCP/IP handshake is done.

In order to calculate a load value, the advisor:

1. Opens a connection with each server.
2. Sends a protocol-specific request message.
3. Listens for a response from the server.
4. Calculates the load value.

After getting the response, the advisor makes an assessment of the server. To calculate this “load” value, most advisors measure the time for the server to respond, and then use this value (in milliseconds) as the load.

You may also set the `connecttimeout` and `receivetimeout` parameters for each advisor. `connecttimeout` is the amount of time the advisor will wait before aborting the connection and `receivetimeout` is the amount of time the advisor will wait before giving up on the data over the socket.

5. Reports the load value to Manager.

If the server does not respond, the advisor returns a negative value (-1) for the load. A downed server is given a weight of zero by the Executor, and packets will not be forwarded to it until the server responds to the advisor again.

Manager obtains the load value reported by the advisor, which is available in the Port column of the Manager report (see Example 5-7 on page 157). The manager obtains these values from all of its sources and sets proportional weight values for Executor.

Customizable advisor settings

An additional feature available since WebSphere Edge Components V5.0 allows you to customize settings for the HTTP and HTTPS advisors. This enables an administrator to set a request type and expected response header per server without writing any code. For example, you can solicit a different request such as GET /index.html and parse the response for the intended header.

The HTTPS advisor fully negotiates an SSL connection and issues the request/response encrypted to the back-end server.

Customizable advisors are an alternative to the custom advisors (this is discussed in “Custom advisors” on page 109) when you don't need the extensive functionality that the custom advisors give you.

Custom advisors

You can also write your own advisors for specific applications. These are called *custom advisors*, and you can write your own advisor based on sample Java code provided with the product. The sample code is available in the <install_path>/servers/samples/CustomAdvisors directory, where <install_path> is the Load Balancer installation path (such as /opt/ibm/edge/lb on AIX, or C:\Program Files\IBM\edge\lb on Windows).

Custom advisors run on the Dispatcher node, and must be written using Java language and compiled with a Java compiler for the Dispatcher machine.

Important: For the Edge Components that are part of IBM WebSphere Application Server Network Deployment V6, you need a Java compiler version 1.4.2.

Class file names must follow the form ADV_<name>.class, where <name> is the name you choose for the advisor.

Using the Java SDK, the compile command is:

```
javac -classpath <install_path>/servers/lib/ibmlb.jar ADV_<name>.java
```

Note: The Load Balancer base classes, found in ibmlb.jar, must be referenced in the classpath during compilation.

The advisor code must then be copied to the <install_path>/servers/lib/CustomAdvisors directory, and it can be started using the command line interface or the graphical interface.

Make sure that Manager is running before you try to start any advisor.

Refer to 5.5, “Load Balancer: additional configuration options” on page 190 for an example of using a custom advisor.

More detailed information about custom advisors, describing how they work, how to write, compile and test them, including examples, development techniques, and interface methods, can be found in the *Load Balancer Administration Guide Version 6.0*, GC31-6858.

4.3.2 Content Based Routing (CBR)

The CBR component load balances based on the content of the request. Load Balancer supports content-based routing in two ways: the CBR component and the Dispatcher CBR forwarding method (discussed in “Forwarding methods” on page 105).

In conjunction with Caching Proxy, the CBR component has the ability to proxy HTTP and HTTPS (SSL) requests to specific servers based on the content requested. The Dispatcher component also provides content-based routing, but it does not require the Caching Proxy to be installed. Because the Dispatcher component's content-based routing is performed in the kernel as packets are received, it can provide faster content-based routing than the CBR component.

When do you use which CBR method?

- ▶ For fully secure SSL traffic (client through server):
 - The CBR component (in conjunction with Caching Proxy) can process SSL encryption/decryption in order to perform content-based routing.
 - The Dispatcher CBR forwarding method can only be configured with SSL ID affinity because it cannot process the encryption/decryption to perform true content-based routing on the requested URL.

- ▶ For HTTP traffic:

The Dispatcher CBR forwarding method provides a faster response to client requests than the CBR component. Also, the Dispatcher CBR forwarding method does not require the installation and use of Caching Proxy.

4.3.3 Site Selector

This component performs load balancing using a DNS round robin approach or a more advanced user-specified approach. Site Selector works in conjunction with a name server to map DNS names to IP addresses. System Metrics (provided by the Metric Server) should be used in addition to advisor weights to achieve a well-balanced and accurate weighting of servers.

4.3.4 Cisco CSS Controller and Nortel Alteon Controller

These controllers can be used to generate server weighting metrics that are then sent to the Cisco and Alteon Switch, respectively, for optimal server selection, load optimization, and fault tolerance.

4.4 Server affinity in Load Balancer

Server affinity is a technique that enables the Load Balancer to remember which balanced server was chosen for a certain client at its initial request. Subsequent requests are then directed to the same server again.

If the affinity feature is disabled when a new TCP/IP connection is received from a client, Load Balancer chooses the right server at that moment and forwards the packet to it. If a subsequent connection comes in from the same client, Load Balancer treats it as an unrelated connection, and again chooses the most appropriate server at that moment.

Server affinity allows load balancing for those applications that need to preserve state across distinct connections from a client. Maintaining state is a requirement of many applications encountered on the Internet today, including “shopping carts,” home banking, and so on.

Some options available to maintain application state based on server affinity are:

- ▶ Stickiness to source IP address
- ▶ Cross port affinity
- ▶ Passive cookie affinity
- ▶ Active cookie affinity
- ▶ URI affinity
- ▶ SSL session ID

The passive cookie, active cookie, and URI affinity options are rules-based. They depend on the content of the client requests.

4.4.1 Stickiness to source IP address

This affinity feature is enabled by configuring the clustered port to be sticky. Configuring a cluster port to be sticky allows subsequent client requests to be directed to the same server. This is done by setting the sticky time to a positive number; the feature can be disabled by setting the sticky time to zero.

The sticky time value represents the timeout of the affinity counter. The affinity counter is reset every time Load Balancer receives a client request. If this counter exceeds sticky time, new connections from this client may be forwarded to a different back-end server.

In Dispatcher and CBR components, you can set the sticky time in three elements of the Load Balancer configuration:

- ▶ **Executor:** setting the sticky time for the Executor makes this value valid for all clusters and ports in the configuration.
- ▶ **Cluster:** you can set a specific sticky time value for each cluster.
- ▶ **Port:** you can set a specific sticky time value for each port.

Important: Setting affinity at the different levels means that any subsequent lower level objects inherit this setting by default (when they are added). In fact, the only true value that is used for sticky time is what is set at the port level. So if you set the sticky time for the Executor to 60, then add a Cluster and Port, these also have a sticky time of 60.

However, if you set a different sticky time for the Cluster or the Port, for example, you set it to 30, then this value overrides the Executor sticky time.

In Site Selector, you set the sticky time on the sitename.

This feature applies to the Dispatcher (all forwarding methods), the CBR and the Site Selector components of Load Balancer.

Note: This affinity strategy has some drawbacks: some ISPs use proxies that collapse many client connections into a small number of source IP addresses. A large number of users who are not part of the session will be connected to the same server. Other proxies use a pool of user IP addresses chosen at random, even for connections from the same user, invalidating the affinity.

4.4.2 Cross port affinity

Cross port affinity is the sticky feature that has been expanded to cover multiple ports. For example, if a client request is first received on one port and the next request is received on another port, cross port affinity allows Dispatcher to send the client requests to the same server.

One example of this feature is a shopping cart application. The user browses the products and adds them to his shopping cart using port 80 (HTTP). When he is ready to place the order, he is redirected to a HTTPS (port 443) site, which will encrypt all communication between the browser and the server. Cross port affinity enables Dispatcher to forward this user's requests for both ports 80 and 443 to the same server.

In order to use this feature, the ports must:

- ▶ Share the same cluster address
- ▶ Share the same servers
- ▶ Have the same sticky time value (not zero)
- ▶ Have the same sticky mask value

More than one port can link to the same cross port. When subsequent connections come in from the same client on the same port or a shared port, the same server will be accessed.

Cross port affinity applies to the MAC and NAT/NAPT forwarding methods of the Dispatcher component.

4.4.3 Passive cookie affinity

Passive cookie affinity is based on the content of cookies (name/value) generated by the HTTP server or by the application server. You must specify a cookie name to be monitored by Load Balancer in order to distinguish which server the request is to be sent to.

If the cookie value in the client request is not found or does not match any of the cookie values of the servers, the most appropriate server at that moment will be chosen by Load Balancer.

This feature applies to both the CBR component and to the Dispatcher's CBR forwarding method.

4.4.4 Active cookie affinity

Active cookie affinity enables load balancing Web traffic with affinity to the same server based on cookies generated by the Load Balancer. This function is enabled by setting the sticky time of a rule to a positive number, and setting the affinity to cookie. The generated cookie contains:

- ▶ The cluster, port, and rule
- ▶ The server that was load balanced to
- ▶ A timeout time stamp for when the affinity is no longer valid

Active cookie affinity formats the cluster/port/server/time information into a key value in the format of IBMCBR#### so the IP and configuration information is not visible to the client browser.

The active cookie affinity feature applies only to the CBR component.

4.4.5 URI affinity

URI affinity allows you to load balance Web traffic to caching proxy servers, which allow unique content to be cached on each individual server. As a result, you will effectively increase the capacity of your site's cache by eliminating redundant caching of content on multiple machines. You can configure URI affinity at the rule level and once it is enabled and the servers are running, then

the Load Balancer will forward new incoming requests with the same URI to the same server.

URI affinity applies to the CBR component and to Dispatcher's CBR forwarding method.

4.4.6 SSL session ID

During establishment of an SSL encrypted session, a handshake protocol is used to negotiate a session ID. This handshaking phase consumes a good deal of CPU power, so directing subsequent HTTPS requests to the same server, using the already established SSL session, saves processing time and increases the overall performance of the HTTP server.

Load Balancer watches the packets during the handshake phase and holds information about the session ID if SSL session negotiation is detected.

The forwarding method used to configure SSL session ID affinity is the Dispatcher's CBR forwarding method.

4.5 Load Balancer topologies

Load Balancer can be configured for different topologies, depending on the number of machines available and the high availability requirements. In this section, we discuss the following topologies:

- ▶ Load Balancer on a dedicated server
- ▶ Collocated servers
- ▶ High availability
- ▶ Mutual high availability

4.5.1 Load Balancer on a dedicated server

This is the basic topology where you install Load Balancer on a dedicated server and configure it to balance the workload between multiple Web servers as shown in Figure 4-5 on page 115.

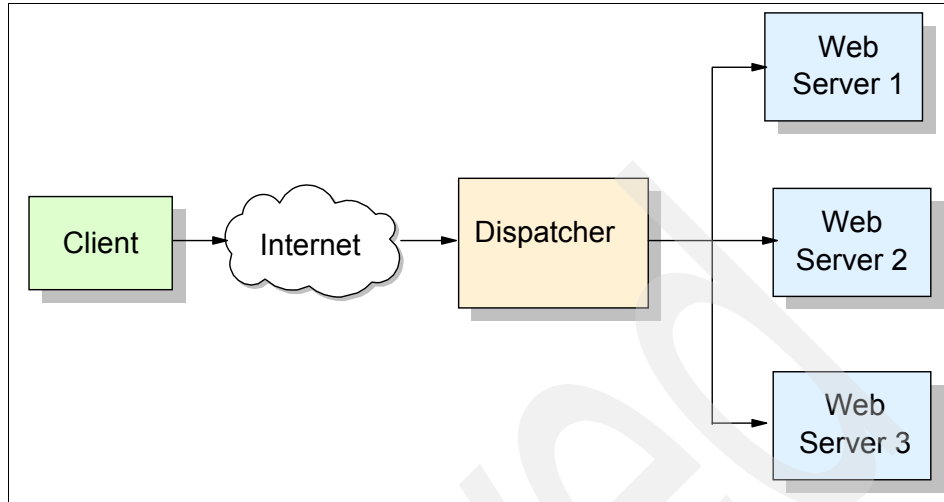


Figure 4-5 Dispatcher basic topology

4.5.2 Collocated servers

Load Balancer can reside on the same machine as a Web server to which it is distributing incoming requests. This is commonly referred to as *collocating* a server. Collocated servers keep initial costs down but allow evolving to a more complex configuration, such as a dedicated server for Dispatcher or even a high availability solution. Collocation is also supported for CBR. See Figure 4-6 on page 116.

Collocation applies to the Dispatcher and Site Selector components.

WebSphere Edge Components V6 added support for collocation on Windows systems for the MAC forwarding method, as mentioned in 4.9, “WebSphere Edge Components V6 new features” on page 125.

Note: To use collocated servers in a Linux environment, you must apply an arp patch which varies for the different kernel versions of Linux. This patch is also necessary for non-collocated balanced servers. For details on what you need to apply, refer to the *Load Balancer Administration Guide Version 6.0*, GC31-6858, available at:

<http://www.ibm.com/software/webservers/appserv/ecinfocenter.html>

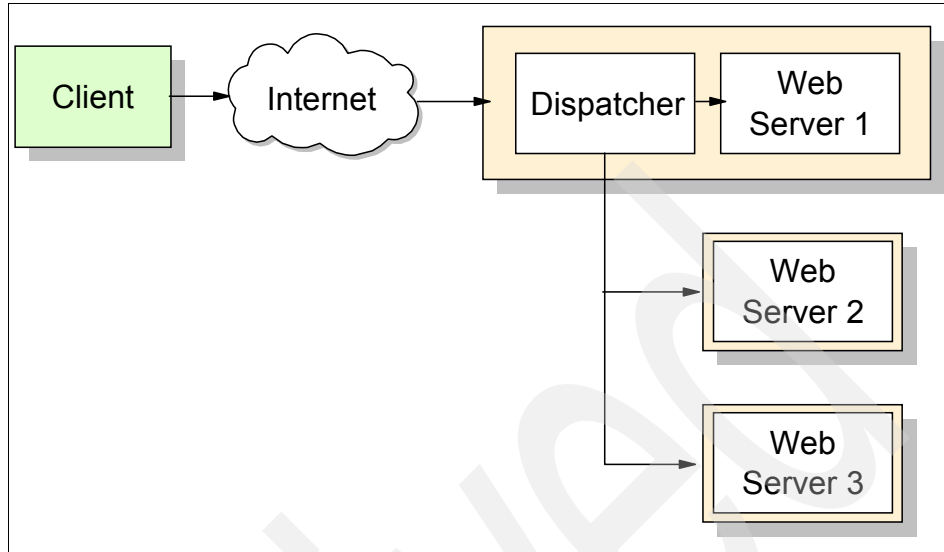


Figure 4-6 Dispatcher collocated topology

Note: A collocated Web server competes for resources with Load Balancer during times of high traffic. However, in the absence of overloaded machines, using collocated servers offers a reduction in the total number of machines necessary to set up a load-balanced site.

4.5.3 High availability

Load Balancer provides a built-in high availability function. It allows you to configure a backup Load Balancer server; if the primary Load Balancer server fails, the backup server will take over the load balancing for all clusters.

The two Load Balancer servers need connectivity to the same clients and to the same cluster of servers, as well as connectivity between themselves. Both Load Balancer servers must be running the same operating systems, and they must be connected to the same network. This configuration is illustrated in Figure 4-7 on page 117.

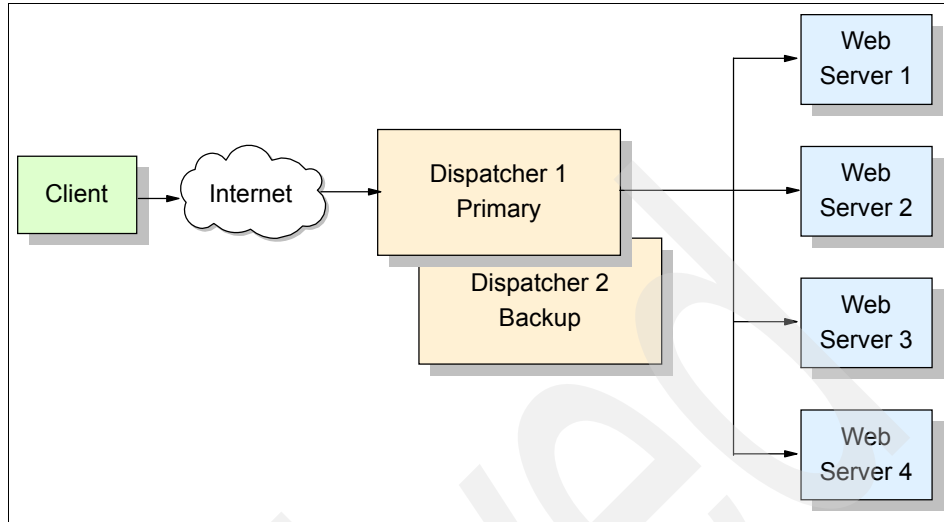


Figure 4-7 Dispatcher high-availability topology

The two Load Balancer servers are referred to as the *primary* server and the *backup* server. These are the *roles* that are associated with each server during the configuration.

Load Balancer servers run in a specific state: one server is *active* and the other server is *standby*. This means that the Load Balancer server that is in active state is the one that is distributing the workload.

The Load Balancer server that is in standby state is monitoring the active one. If the active server fails, the standby server performs a failover; it switches to active state and starts load balancing the cluster. So the state of a server changes when a failure occurs, but the roles do not change during a failure.

Note: You can also configure the backup server to respond to other clusters' addresses while it works as a standby server for the clusters on the primary server. See 4.5.4, "Mutual high availability" on page 118.

When the primary server is operational again, but in standby state, you can either decide that it automatically forces a takeover and it becomes the active server again, or it stays in standby mode, monitoring the other server for a failure. In this last case, you will have to manually force a takeover if you later want it to become the active server again. This is known as auto recovery or manual recovery.

In order to monitor the health of the active server, heartbeats are sent every half second. The failover occurs when the standby server receives no response from

the active server within two seconds. Another possible reason for a failover is when the standby server is able to ping more reach targets than the primary machine. A reach target is a server out of the load balancing environment that is used for an external connectivity test. We usually recommend using the default gateway of the network as the reach target of your configuration.

Note: By default, high availability is only supported for the Dispatcher component, not for the Site Selector.

For a highly available Site Selector, you should configure Dispatcher to host a cluster in front of multiple Site Selectors. The cluster will act as the sitename DNS clients are requesting. The Dispatcher tier can provide the failover mechanism and the Site Selectors will then have no single point of failure. The DNS advisor to be used to detect a failed site selector is included with the product.

4.5.4 Mutual high availability

The mutual high availability feature of Load Balancer provides the administrator with the opportunity to use both Dispatcher servers in a high availability configuration, where both servers actively perform load balancing for a cluster and are the backup for each other. So, in a simple high-availability configuration, only one machine performs load balancing. In mutual high availability, both machines participate in load balancing.

For mutual high availability, client traffic is assigned to each Dispatcher server on a cluster address basis. As illustrated in Figure 4-8 on page 119, you have to configure a cluster for each node that is a primary Dispatcher (so you have at least two clusters in your environment). Each cluster can be configured with the NFA (non-forwarding address) of its primary Dispatcher. The primary Dispatcher server normally performs load balancing for that cluster. In the event of a failure, the other machine performs load balancing for both its own cluster and for the failed Dispatcher's cluster.

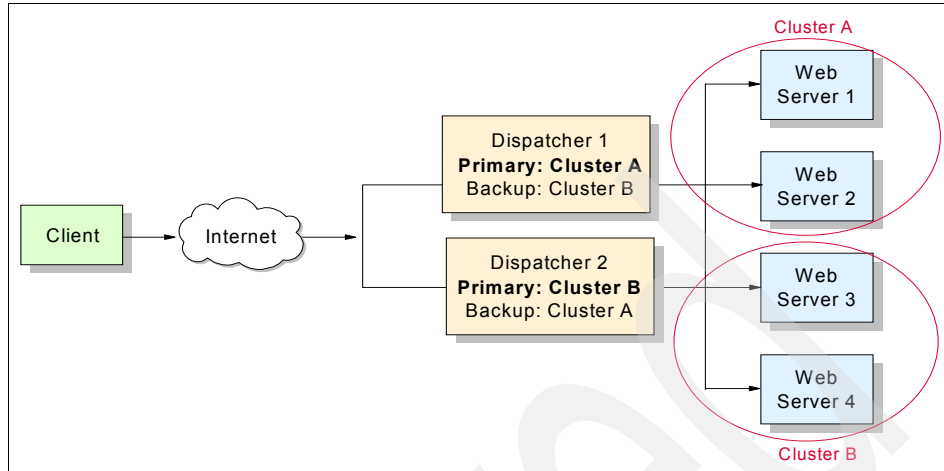


Figure 4-8 Dispatcher mutual high availability topology

4.6 Dispatcher scripts

Dispatcher scripts are command-based scripts that are automatically run whenever a certain event is triggered. They are mostly used in high availability environments.

The following is a list of some of the scripts that Dispatcher uses:

- ▶ **goActive**
This script is run every time a Load Balancer server switches into active state.
- ▶ **goStandby**
This script is run every time a Load Balancer server goes into standby state.
- ▶ **goInOp**
This script is run when the Load Balancer Executor component is stopped or when it is started.
- ▶ **goldle**
This script is not used in high availability environments. It is run when the Load Balancer server goes into idle state (when the machine is a stand-alone dispatcher, or when the high availability features have not yet been added). Do not use this script if you are configuring high availability.

► **serverDown**

This script is run when the Manager identifies that a balanced server is down. Using this script is not mandatory, but you can use it to send an e-mail message to the administrator or log the information to a file.

► **serverUp**

This script is run when the Manager identifies that a balanced server is back up. This script is also not mandatory, you can use it to send an e-mail message to the administrator or log the information to a file.

► **highavailChange**

This script is run when the state of a Load Balancer server changes or when the Executor is started. It is not mandatory, but you can use it to send an e-mail message to the administrator or log the information to a file.

These scripts must be placed in the <product_install_path>/servers/bin directory. In UNIX systems, you need to add execution permission to these files.

You will find a set of examples in the <product_install_path>/servers/samples directory. You can start building your own scripts based on these examples. If you want to use the samples, copy them to the bin subdirectory, rename them and edit them there.

There are two sets of samples for the high availability scripts (goActive, goStandby and goInOp): one for regular high availability and one for mutual high availability.

We show how to use some of these scripts in 5.3.4, “Configuring the high availability scripts” on page 171.

4.7 Load Balancer features comparison

Table 4-1 on page 121 compares most of the characteristics of each forwarding method and CBR component.

You can use this table to check which features you need and what forwarding method or component can provide them to you.

Note that the columns containing the components are ordered according to how fast the response time of each component is, from MAC forwarding (which has the shortest response time) to CBR component (which has the longest response time).

Table 4-1 Load Balancer features comparison table

	MAC forwarding	NAT forwarding	CBR forwarding	CBR component (requires CP) ^a
High availability	Yes	Yes	Yes (does not fail over content information)	Not internal, it needs the Dispatcher component
Local routing	Yes	Yes	Yes	Yes
Remote routing	No (except with advanced wide area lb config.)	Yes	Yes	Yes
Routing based on content	No	No	Yes (except HTTPS)	Yes (including HTTPS)
Affinity based on IP address	Yes	Yes	Yes	Yes
Passive cookie affinity	No	No	Yes	Yes
Active cookie affinity	No	No	No	Yes
URI affinity	No	No	Yes	Yes
Session affinity	No	No	Yes (HTTP only)	Yes (HTTP and HTTPS)
SSL session ID affinity	No	No	Yes (no decryption)	No
Cross port affinity	Yes	Yes	No (except with passive cookie affinity with global cookie values)	No
Caching	No	No	No	Yes

a. CP = Caching Proxy

For more information about the features listed in this table, refer to the previous topics in this chapter and to the product documentation, available at:

<http://www.ibm.com/software/webservers/appserv/ecinfocenter.html>

4.8 Caching Proxy overview

Caching Proxy intercepts requests from the client, retrieves the requested information from the content-hosting machines, and delivers that information back to the client. You can configure Caching Proxy to handle protocols such as HTTP, FTP, and Gopher.

The Caching Proxy stores cachable content in a local cache before delivering it to the requestor. Examples of cachable content include static Web pages and whole dynamic Web pages. Caching enables the Caching Proxy to satisfy subsequent requests for the same content by delivering it directly from the local cache, which is much quicker than retrieving it again from the content host.

The Caching Proxy can be configured as a Reverse or Forward Proxy Server. The cache can be stored on physical storage devices or in memory.

4.8.1 Forward proxy

The Caching Proxy can be configured as a *forward proxy*. Normally, Internet access providers configure it in this mode.

When configured in forward proxy mode, it handles requests from multiple client browsers, retrieves data from the Internet and caches the retrieved data for future use. In this case, the client browser has to be configured to use the proxy server.

When a client requests a page, the caching proxy connects to the content host located across the Internet, sends the request that it received from the client, then caches the retrieved data and delivers the retrieved data to the client. If other client sends the same request afterwards, it will be served from the cache. This decreases the network usage and provides better response times.

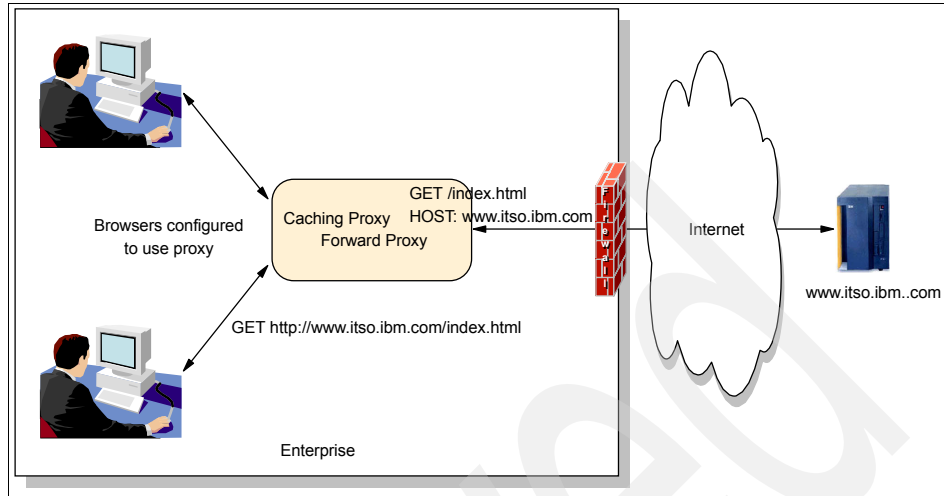


Figure 4-9 Forward proxy scenario

4.8.2 Reverse proxy (IP forwarding)

IP-forwarding topologies use a *reverse proxy* server, such as the Caching Proxy, to receive incoming HTTP requests and forward them to a Web server. The Web server forwards the requests to the application servers for actual processing. The reverse proxy returns completed requests to the client, masquerading the originating Web server.

If a client then requests the same data the next time, it will not be sent to the back-end server for processing, but instead will be served from the cache. This prevents unnecessary back-end processing for the same data requests, thus providing better response times.

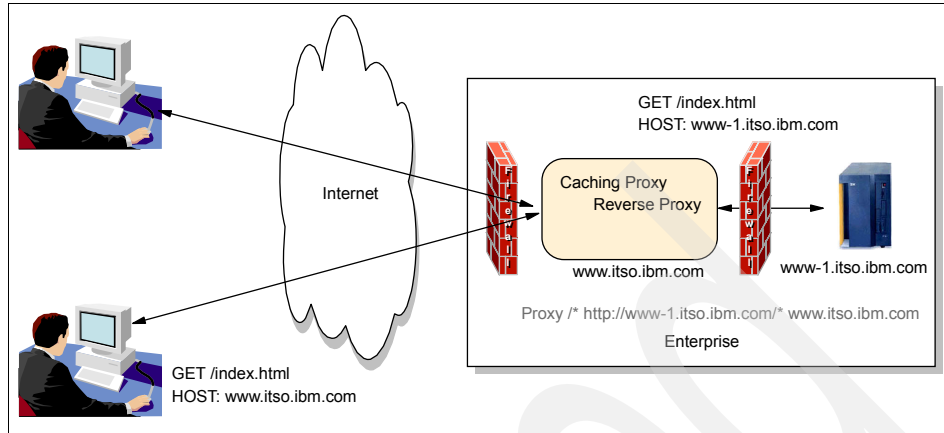


Figure 4-10 Reverse proxy scenario

4.8.3 Using multiple Caching Proxy servers

Multiple caching proxy servers can be configured to increase your site performance, compared with a single caching proxy at peak load times. The Load Balancer Dispatcher component can be used to distribute the load to the caching proxy servers.

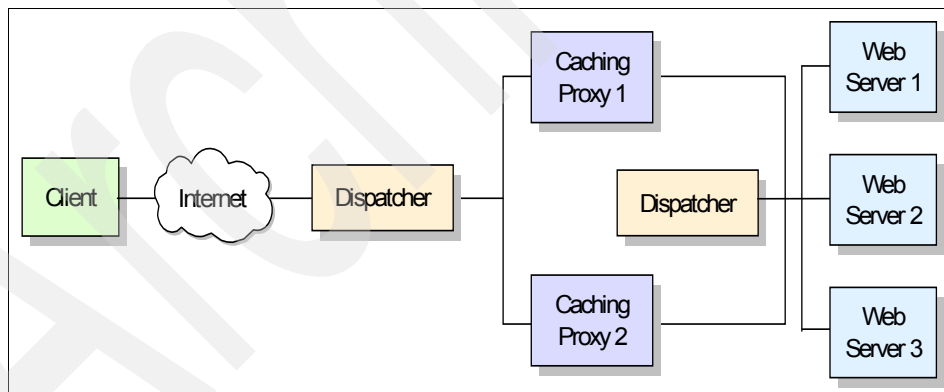


Figure 4-11 Load balancing multiple caching proxy servers

4.8.4 Dynamic caching

The dynamic caching function enables the Caching Proxy to cache dynamically generated content in the form of responses from JSPs and servlets generated by an IBM WebSphere Application Server. A caching proxy adapter module is used at the application server to modify the responses, so that they can be cached at

the proxy server in addition to being cached in the application server's dynamic cache. With this feature, dynamically generated content can be cached at the entry point of the network, avoiding repeated requests to the application server, when the same content is requested by multiple clients. However, the Caching Proxy can only cache full pages, not fragments, and all subcomponents of that page must also be cachable. Secure content requiring authorization is not cached externally at the proxy server.

See Chapter 10, "Dynamic caching" on page 501 for more information about this topic and for details on how to configure the Caching Proxy to cache dynamic content.

4.9 WebSphere Edge Components V6 new features

WebSphere Edge Components V6 provide several new features and modifications:

- ▶ The Load Balancer Dispatcher now supports collocation on Windows systems for the MAC forwarding method. In previous versions, collocation on Windows systems was only available for NAT/NAPT and CBR forwarding methods.
- ▶ The Load Balancer Dispatcher now offers a **stop** command for the Executor component on Windows systems. In previous versions, this command was only available on UNIX platforms.
- ▶ The Load Balancer Dispatcher provides a new method for connection cleanup, which improves the performance of connection record allocation and reuse. It no longer needs the **dscontrol executor set fincount** command, so this command has been deprecated and removed from the product.
- ▶ Load Balancer Dispatcher **dsconfig** and **ndconfig** commands have been deprecated. Make sure you replace these commands in all existing scripts with their replacement command: **dscontrol executor configure**.
- ▶ Previous versions supported both **ndcontrol** and **dscontrol** commands. Now only the **dscontrol** command is supported. Make sure you update all scripts.
- ▶ Load Balancer now requires the 32-bit Java 2 Runtime Environment version 1.4.2.
- ▶ WebSphere Edge Components added support for AIX V5.3, Linux for iSeries and Linux for pSeries (the previous version only supported Linux for Intel and Linux for S/390® zSeries). For more information, refer to *Concepts, Planning, and Installation for Edge Components Version 6.0*, GC31-6855.
- ▶ In UNIX systems, the default browser for viewing the documentation is now Mozilla 1.4 or higher, and it is required to run the LaunchPad tool. On

Windows systems, the recommended browsers are Internet Explorer 5.5 or higher and Mozilla 1.4 or higher.

Using IBM WebSphere Edge Components

In this chapter, we describe how to configure and set up various usage examples for the Load Balancer and the Caching Proxy components of IBM WebSphere Edge Components.

For WebSphere Load Balancer, we explain how to configure the most commonly used scenarios, such as:

- ▶ Basic load balancing scenario
- ▶ High availability scenario
- ▶ NAT scenario
- ▶ How to use sample custom advisors

We do not intend to exhaust all possibilities, so if you need information about functionality that is not covered here, please refer to the *Load Balancer Administration Guide Version 6.0*, GC31-6858 or contact your IBM Support Center in order to get more information about a specific implementation.

For the Caching Proxy, we cover the following tasks:

- ▶ Caching Proxy installation
- ▶ Caching Proxy configuration
- ▶ Managing the Caching Proxy process

5.1 Load Balancer installation

You can install the WebSphere Edge Components product using either the common installation wizard or the operating system tools and commands.

We first describe the installation on a Windows 2000 server using the wizard, and later we describe the installation on an AIX server using SMIT (a management tool provided by the operating system). See 5.1.1, “Load Balancer installation wizard” on page 128 and 5.1.2, “Load Balancer installation using SMIT in AIX” on page 132.

Before starting the installation, refer to *Load Balancer Administration Guide Version 6.0*, GC31-6858 for the prerequisites and supported operating systems.

5.1.1 Load Balancer installation wizard

The Edge components installation media provides an installation wizard for all platforms so the installation is similar for all supported operating systems.

Important: Before starting with the installation, you should have Java Runtime (V1.4.2 or later) installed on your system.

1. Mount the installation media and start LaunchPad by running `launchpad.sh` (in UNIX/Linux systems) or `launchpad.bat` (in Windows systems).

The LaunchPad window opens as shown in Figure 5-1 on page 129.

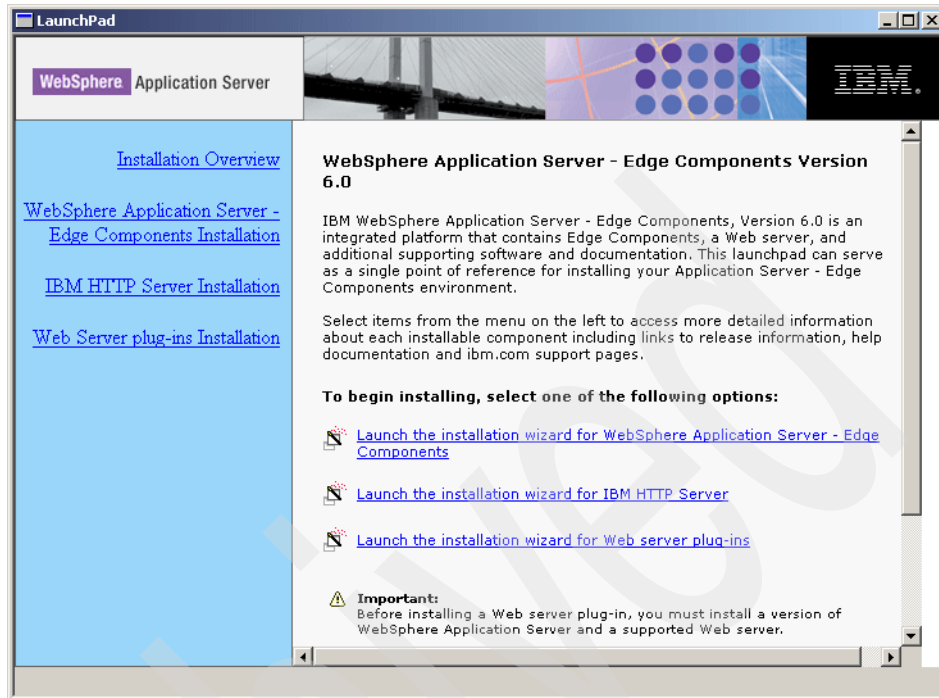


Figure 5-1 LaunchPad window

Click **Launch the installation wizard for WebSphere Application Server - Edge Components**.

2. Click **Next** on the Welcome screen and click **Yes** to accept the product license.
3. In the Component Selection window, you can select which components you want to install. Select the **Load Balancer** checkbox, and click the **Change Subcomponents** button as shown in Figure 5-2 on page 130.

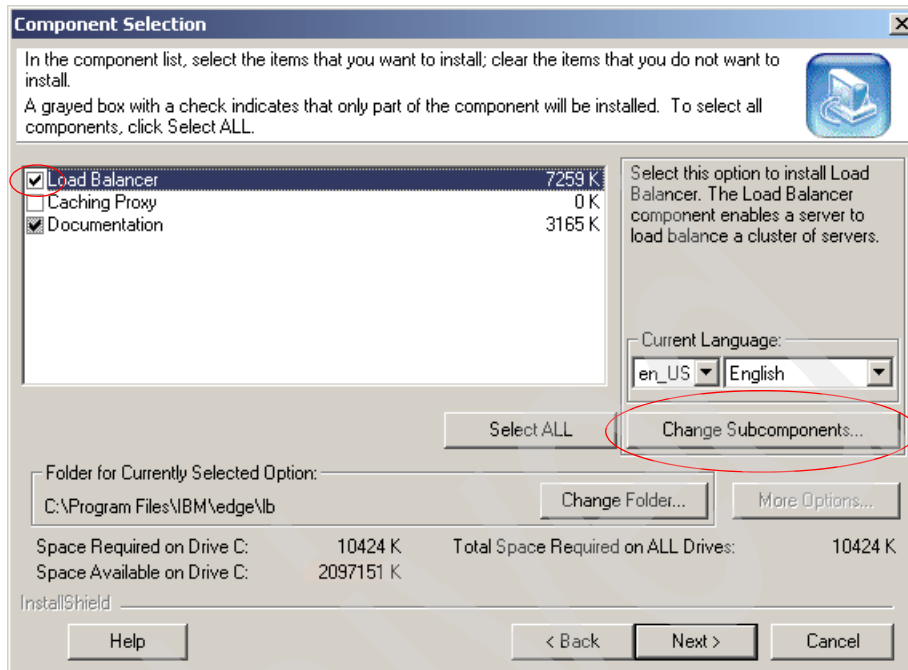


Figure 5-2 Component Selection window

4. The Subcomponent Selection window opens. Select the subcomponents you want to install. The Administration and License subcomponents are mandatory. By default, all subcomponents are selected, as shown in Figure 5-3 on page 131. Click **OK** to return to the Component Selection window.

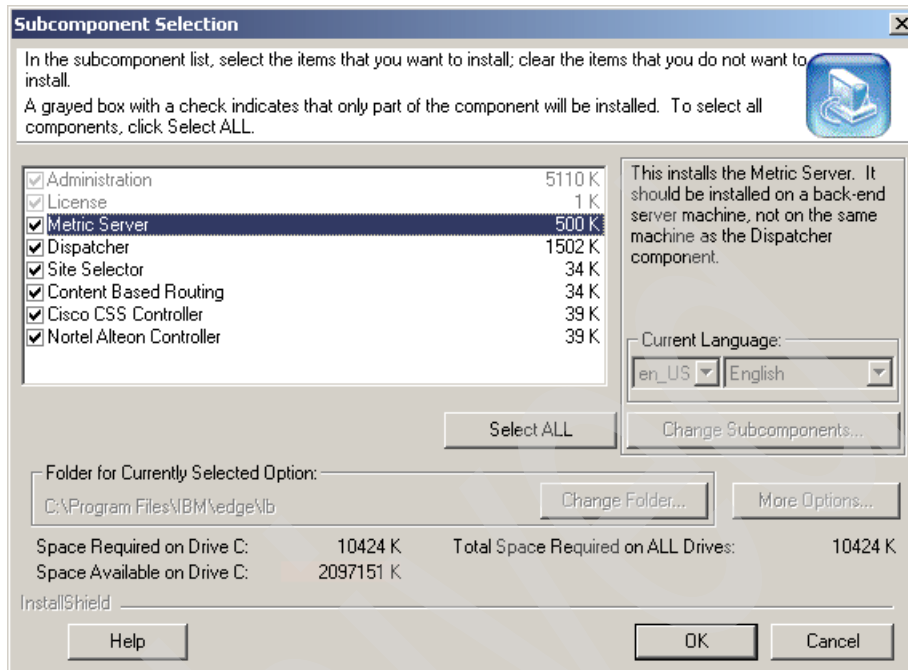


Figure 5-3 Subcomponent Selection window

5. The default installation path is C:\Program Files\IBM\edge\lb for Windows systems (for UNIX/Linux systems, it is /opt/ibm/edge/lb). If you want to install the product in a different directory, click **Change Folder** and enter the correct path. Click **Next** to continue the installation.
6. Verify that the selected options are listed in the Installation Selection Summary, and click **Finish** to start the installation, as shown in Figure 5-4 on page 132.

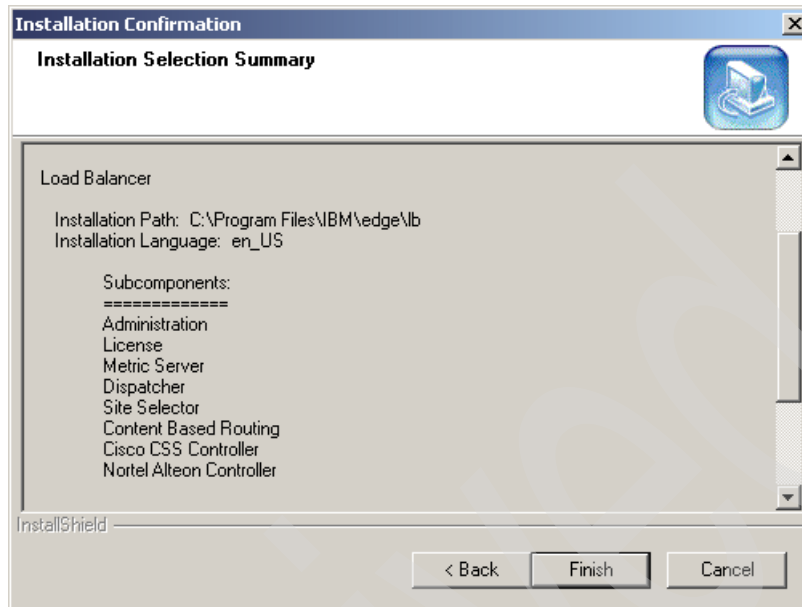


Figure 5-4 Installation confirmation window

7. At the end of the installation, you have the option to reboot the server. Make sure you do so before using the product.

5.1.2 Load Balancer installation using SMIT in AIX

AIX provides a tool to manage the operating system, which is SMIT. In this section, we describe how to install WebSphere Edge Components using SMIT.

1. Log on as the root user.
2. Mount the WebSphere Edge Components installation media and change to the directory you used as the mount point (for example, /cdrom).
3. Change to the lb directory:


```
cd lb
```
4. Run the following command:


```
smit install_all
```
5. On the Install and Update from ALL Available Software screen, type the full path to the lb directory mentioned in step 3 or type a period (.) which represents the current directory, as shown in Figure 5-5 on page 133.

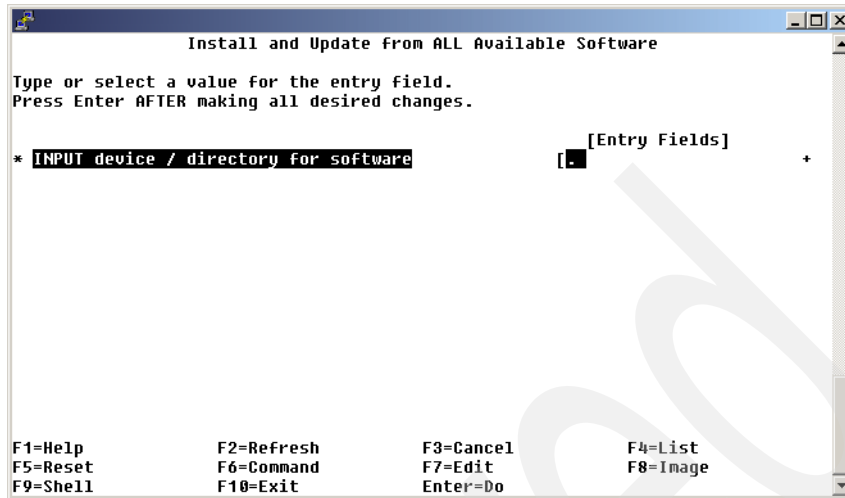


Figure 5-5 Selecting the path of the installation media

6. On the Install and Update from ALL Available Software screen, put the cursor into the SOFTWARE to install field (see Figure 5-6 on page 134) and press the **F4** key (**ESC+4** in VT100 and ASCII emulators).
7. From the SOFTWARE to install list, select the components you want to install. We selected the following components:

- LB Admin Messages - U.S. English (ibmlb.msg.en_US.admin.rte)
- LB Administration (ibmlb.admin.rte)
- LB Base (ibmlb.base.rte)
- LB Dispatcher (ibmlb.disp.rte)
- LB Dispatcher Device Driver (ibmlb.lb.driver)
- LB Documentation (ibmlb.doc.rte)
- LB License (ibmlb.lb.license)
- LB Messages - U.S. English (ibmlb.msg.en_US.lb.rte)
- Load Balancer Documentation - U.S. English (ibmlb.msg.en_US.doc)

You may need to scroll the screen to the left using the right arrow key in order to see the fileset names listed in parenthesis.

After you finish selecting the filesets, press the **Enter** key.

8. Back on the Install and Update from ALL Available Software screen, select **yes** for the *ACCEPT new license agreements?* field as shown in Figure 5-6 on page 134, and press **Enter** to start the installation.

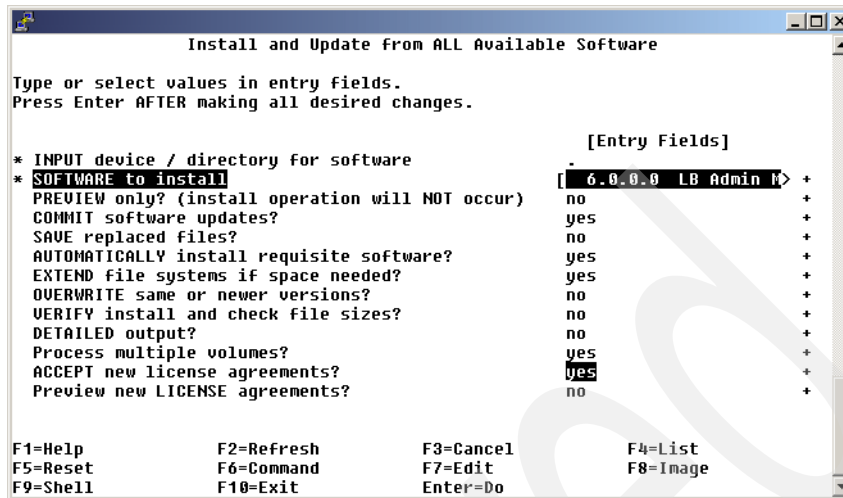


Figure 5-6 SMIT installation screen

9. When the installation finishes, check the installation summary to make sure that all filesets were installed successfully, as shown in Example 5-1.

Example 5-1 Installation summary

Installation Summary

Name	Level	Part	Event	Result
ibm1b.lb.license	6.0.0.0	USR	APPLY	SUCCESS
ibm1b.doc.rte	6.0.0.0	USR	APPLY	SUCCESS
ibm1b.admin.rte	6.0.0.0	USR	APPLY	SUCCESS
ibm1b.base.rte	6.0.0.0	USR	APPLY	SUCCESS
ibm1b.lb.driver	6.0.0.0	USR	APPLY	SUCCESS
ibm1b.msg.en_US.doc	6.0.0.0	USR	APPLY	SUCCESS
ibm1b.msg.en_US.admin.rte	6.0.0.0	USR	APPLY	SUCCESS
ibm1b.msg.en_US.lb.rte	6.0.0.0	USR	APPLY	SUCCESS
ibm1b.disp.rte	6.0.0.0	USR	APPLY	SUCCESS

5.1.3 Post installation tasks

Before trying to start any Load Balancer components, it is necessary to check for a supported Java 2 Runtime Environment available to the operating system. Refer to *Concepts, Planning, and Installation for Edge Components Version 6.0*, GC31-6855 for all software requisites.

For the AIX V5.2 operating system, WebSphere Edge Components V6 requires 32-bit Java 2 Runtime Environment version 1.4.2. We downloaded it from the following URL:

<http://www.ibm.com/developerworks/java/jdk/index.html>

We installed it according to the instructions provided with the package and added the /usr/java14/bin directory to the contents of the PATH environment variable for user root, by adding the following commands to the \$HOME/.profile file:

```
PATH=/usr/java14/bin:$PATH
export PATH
```

After that, we ran the command shown in Example 5-2 to make sure the default Java command being used is the correct one.

Example 5-2 Checking the Java command version

```
# java -version
java version "1.4.2"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2)
Classic VM (build 1.4.2, J2RE 1.4.2 IBM AIX build ca1420-20040626 (JIT enabled:
jitc))
```

5.2 Load Balancer configuration: basic scenario

This scenario represents the most simple example of load balancing where Load Balancer is configured on one system only and load balances the traffic between two Web servers, as shown in Figure 5-7.

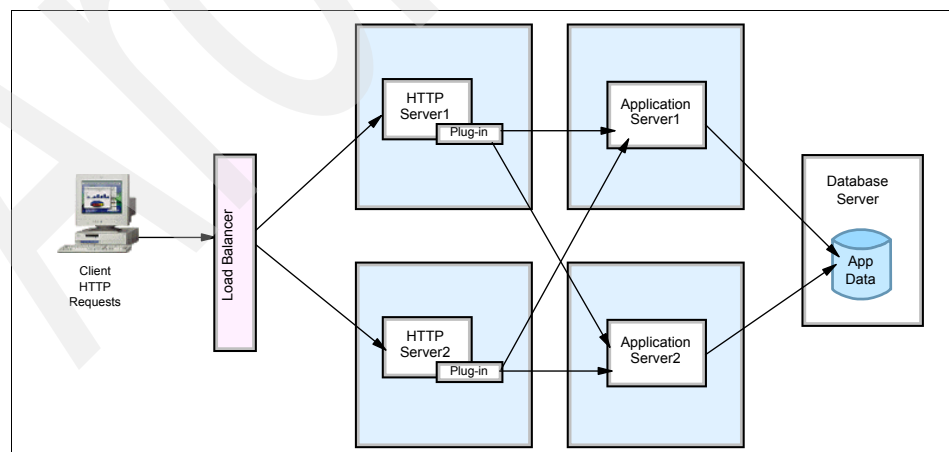


Figure 5-7 Basic Load Balancer scenario

This scenario shows the Dispatcher component using the MAC forwarding method. It does not have high availability, which will be added in the next scenario (see “Load Balancer: high availability scenario” on page 162).

The steps to implement this scenario are described in:

- ▶ 5.2.1, “Configuring the Load Balancer cluster” on page 136
- ▶ 5.2.2, “Configuring the balanced servers” on page 148
- ▶ 5.2.3, “Testing the basic scenario” on page 156

5.2.1 Configuring the Load Balancer cluster

The configuration can be done using the Load Balancer graphical user interface (**lbadmin**) or using the command line interface (**dscontrol**). We first explain how to do it using the GUI, and later we show the commands (which give you the same result).

In order to send commands through the GUI or through the command line interface to Load Balancer, you need to start the component element that receives those commands and executes them.

In this scenario, we only use the Dispatcher component.

1. Start the Dispatcher server in order to start configuring it. To do so, run the following command:

```
dsserver
```

2. Open the Load Balancer GUI by running the following command:

```
lbadmin
```

The Load Balancer GUI is a Java client that can also be installed on a client machine, so the administrator can work remotely.

3. When the Load Balancer administration tool comes up, right-click **Dispatcher** in the left pane and select **Connect to Host...** as shown in Figure 5-8 on page 137.

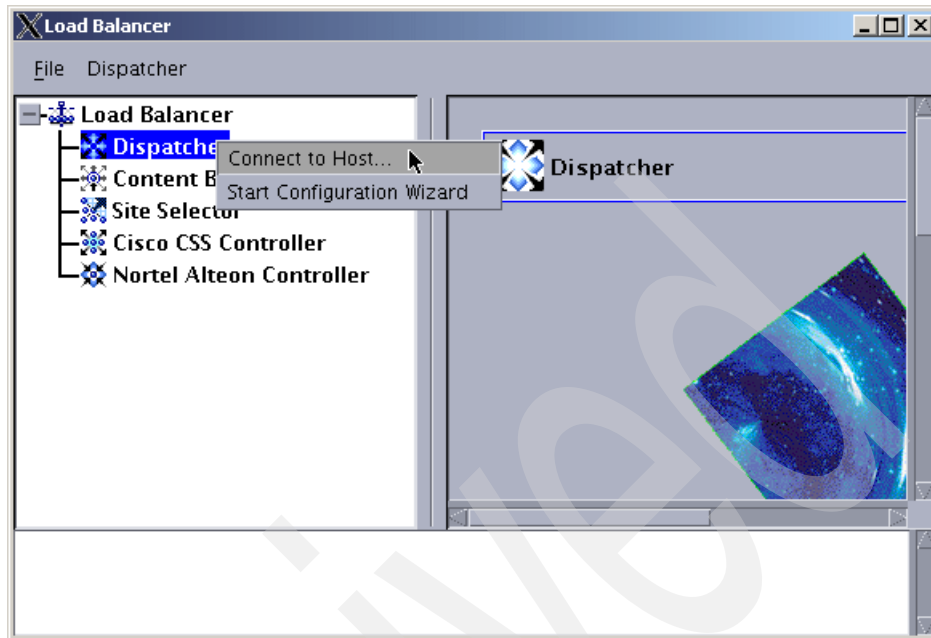


Figure 5-8 Load Balancer administration tool

4. A pop-up window is displayed, prompting you for the Load Balancer server which you want to connect to. Select the hostname of the Load Balancer server, as shown in Figure 5-9.

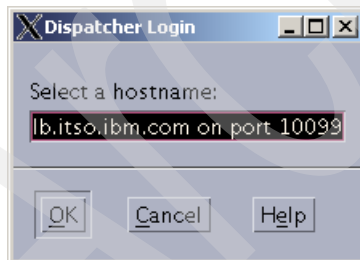


Figure 5-9 Selecting the Load Balancer server

After connecting to the Load Balancer server, a new entry is added to the GUI window in the left pane, containing the hostname of the selected server. All the configuration we perform from now on is added to this element in a tree structure.

5. Now we need to start the Executor component, which is the component that actually distributes the load to the servers. Right-click **Host:lb.itso.ibm.com** and select **Start Executor**, as shown in Figure 5-10 on page 138.

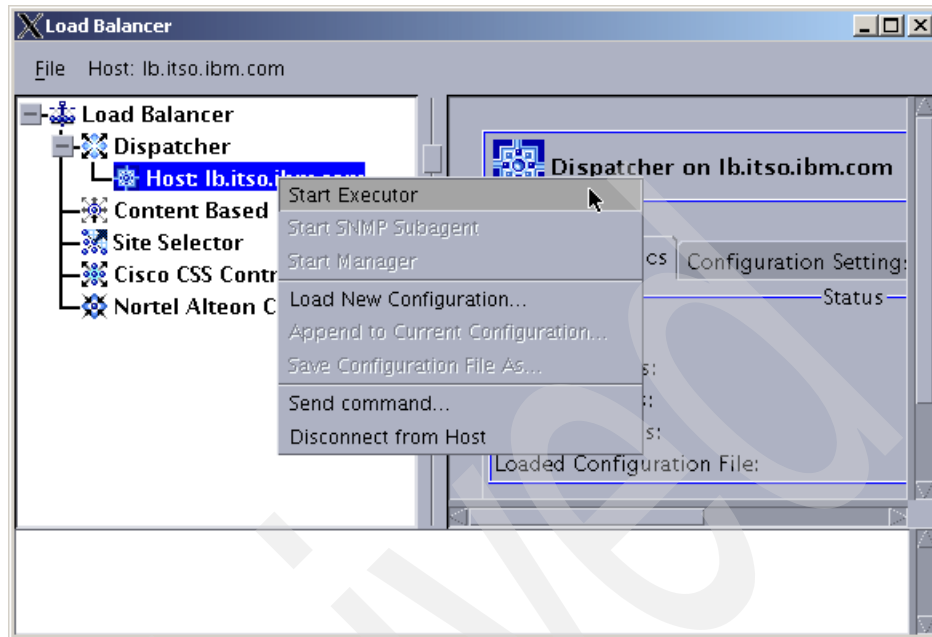


Figure 5-10 Starting Executor

If Executor is started successfully, a new item named Executor is added to the left pane. In our scenario, the Load Balancer IP address is 10.20.10.102, so this IP address is shown in this new item as well.

Tip: For every action you perform, you can see a message in the bottom pane of the GUI window that confirms whether or not the action was performed successfully.

6. The next thing we need to do is to add our cluster. In our scenario, we have a cluster called cluster.itso.ibm.com (10.20.10.100) and this cluster contains two Web servers, http1 (10.20.10.103) and http2 (10.20.10.104).

Right-click **Executor: 10.20.10.102** and select **Add Cluster...**, as shown in Figure 5-11 on page 139.

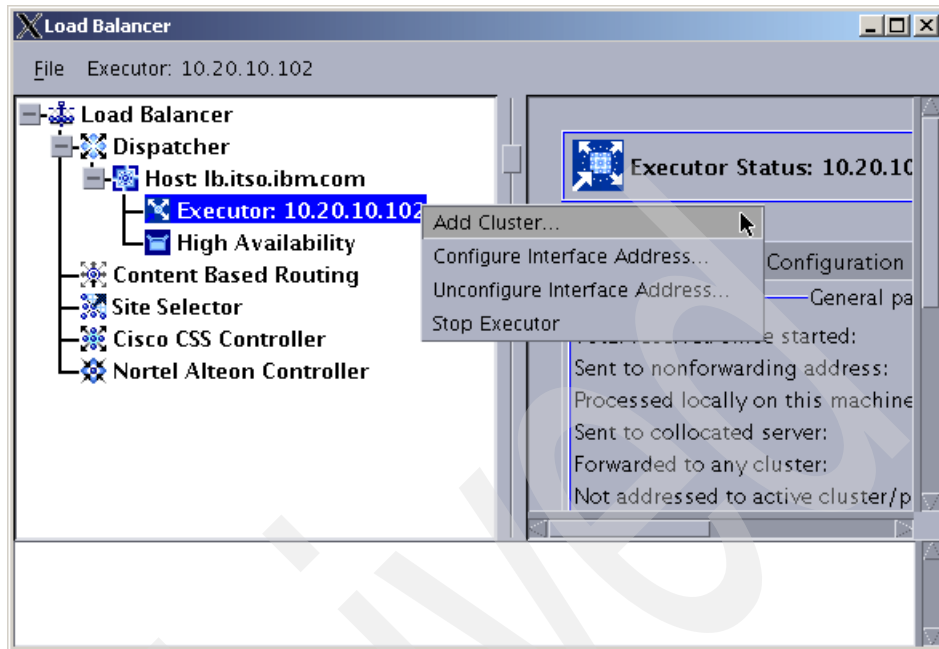


Figure 5-11 Adding a cluster

7. A new window is displayed, prompting for the necessary information to add the new cluster. Type the name of the cluster in the Cluster field (we recommend using the hostname). Then type the cluster IP address in the Cluster address field, and make sure that the Load Balancer's IP address is selected in the Primary host for the cluster field.

Check the option **Configure this cluster?** as shown in Figure 5-12 on page 140. This option is used to create an IP alias in the operating system for the cluster IP address. You can also uncheck this option and add the IP alias manually using operating system tools or commands.

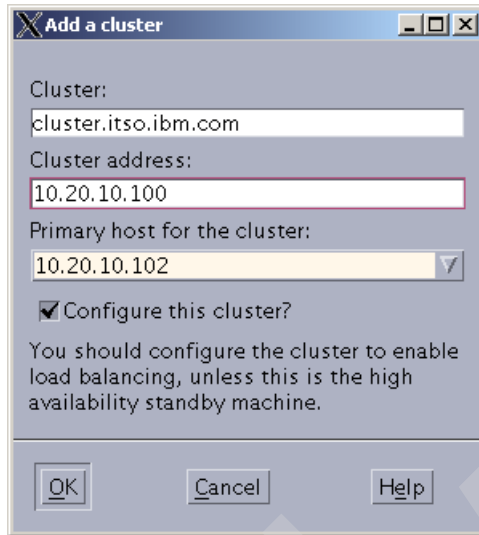


Figure 5-12 Filling in the information to add a cluster

8. If you checked the **Configure this cluster?** checkbox, another window is displayed. Enter the interface identification in the Interface name field (in our server the interface that is associated with the IP address 10.20.10.102 is en0) and the network mask in the Netmask field, as shown in Figure 5-13.

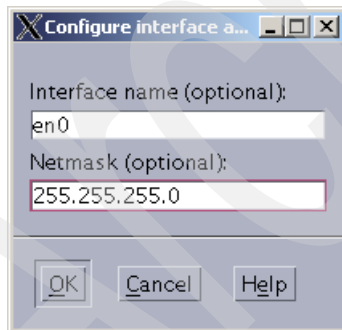


Figure 5-13 Configuring the interface

Although these fields are optional, we recommend you provide them, otherwise Load Balancer uses the default values, which may not be correct for your system. For example, the default network mask for our network (10.20.10.0) is 255.0.0.0 (because it is a class A network), but we use a subnet mask. So the default value would not work properly in this example.

Note: In our tests using a Windows machine, the interface was automatically configured as en1 when we left the fields unfilled, which resulted in errors.

If you have only one Ethernet card in your machine, the interface name will be en0. Likewise, if you have only one Token Ring card, the interface name will be tr0. If you have multiple cards of either type, you will need to determine the mapping of the cards. Use the following steps:

Click **Start -> Run** and run **regedit**. Expand **HKEY_LOCAL_MACHINE -> Software -> Microsoft -> Windows NT® -> Current Version -> NetworkCards**.

The network interface adapters are listed under Network Cards. Click each one to determine the interface type. The type of interface is listed in the Description column. The names assigned by the **executor configure** command map to the interface types. For example, the first Ethernet interface in the list is assigned to en0, the second to en1, and so on; the first Token Ring interface is assigned to tr0, the second to tr1, and so on.

A new item which identifies your cluster is added to the left pane of the GUI.

9. Add each port that will be load balanced by Dispatcher. Right-click **Cluster: cluster.itso.ibm.com** and select **Add Port....**

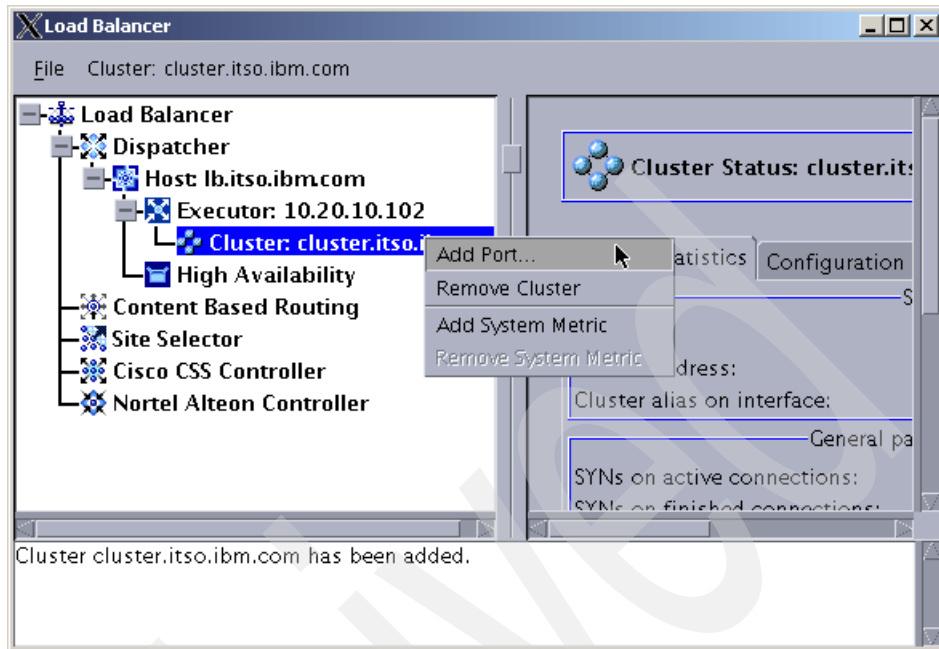


Figure 5-14 Adding a port

The port that we are adding refers to the port that the client will access. In our scenario, we use port 80.

Fill in the number of the port in the Port number field and select **MAC Based Forwarding** in the Forwarding method field, as shown in Figure 5-15.

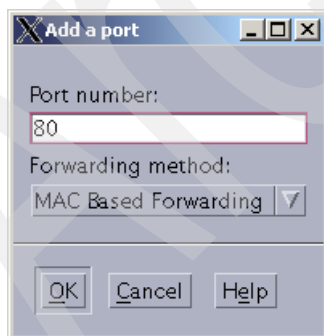


Figure 5-15 Port information

For more information about the available forwarding methods, refer to “Forwarding methods” on page 105.

A new item representing port 80 is added to the left pane of the GUI.

10. Add the servers that will receive the load for port 80 of cluster cluster.itso.ibm.com. Right-click **Port:80** and select **Add Server...**, as shown in Figure 5-16.

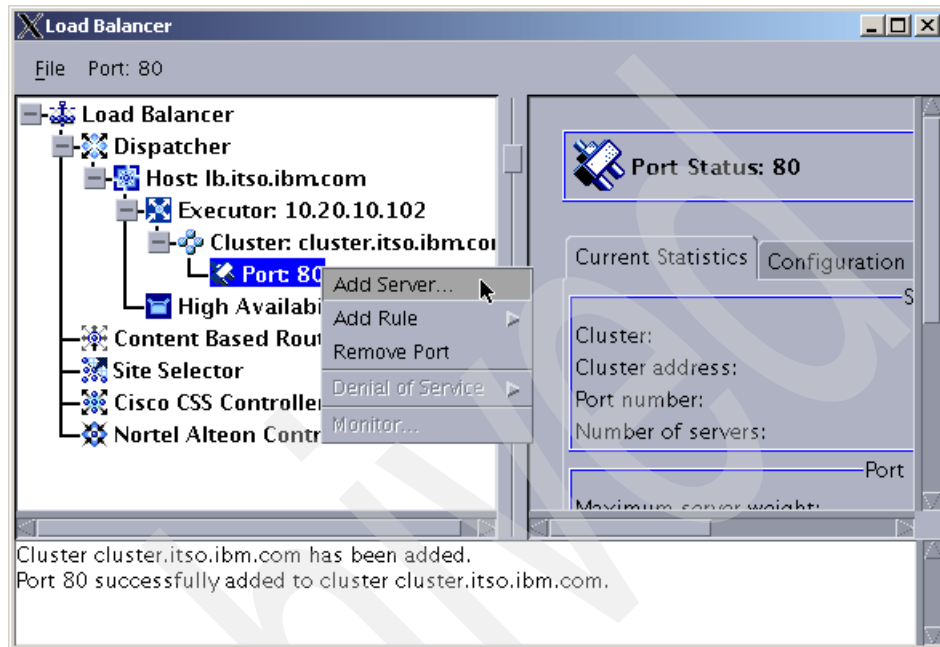


Figure 5-16 Adding a server

The next window prompts you for the information of the first server. Fill in the hostname of your Web server in the Server field and enter its IP address in the Server address field, as shown in Figure 5-17 on page 144.

The first server we add in our scenario is http1, and its IP address is 10.20.10.103.



Figure 5-17 Adding the first balanced server

Note that the *Network router address* checkbox is disabled because we selected MAC Based Forwarding and this forwarding method does not allow load balancing to remote servers.

Repeat step 10 on page 143 for all servers in the cluster.

We also add our second server; the hostname of this server is http2 and the IP address is 10.20.10.104.

The load balancing part of the configuration is done. All the information that Dispatcher needs to provide load balancing for our cluster is now configured. But we also need the Manager component because we want to work with dynamic weight values and failure detection.

11. Therefore, we now need to start the Manager component. Right-click **Host: lb.itso.ibm.com** and select **Start Manager**, as shown in Figure 5-18 on page 145.

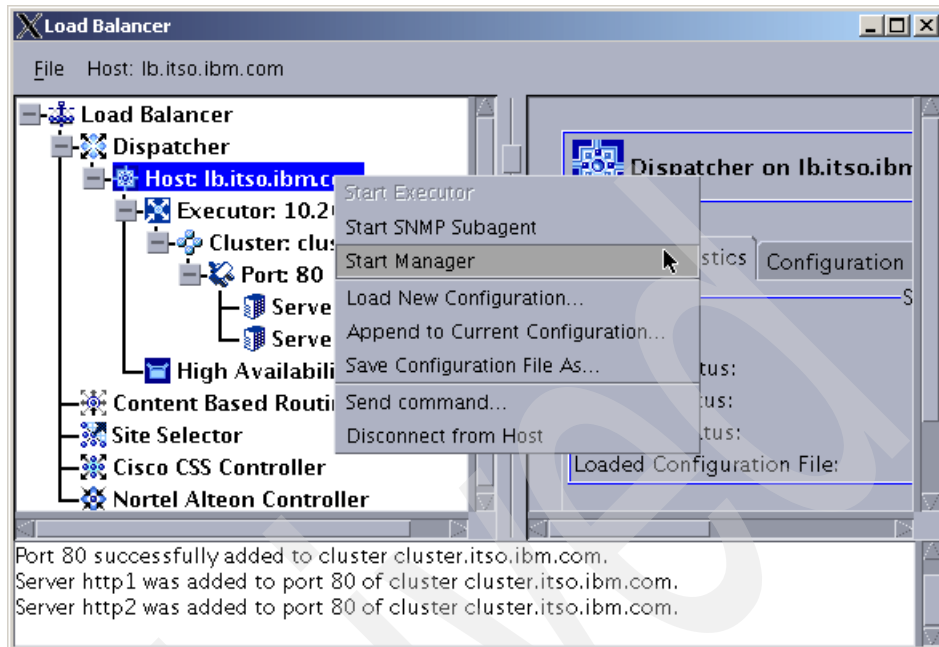


Figure 5-18 Starting Manager

A window is displayed in which you can select the name of the Manager log file and the metric port, as shown in Figure 5-19. We chose the default options.

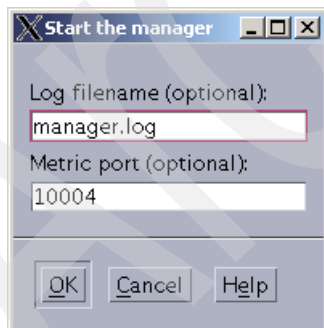


Figure 5-19 Manager options

Manager needs advisors in order to generate a weight value based on the response time from each server in the cluster. The advisor is also needed in order to detect a failure in the service of any balanced server (in our case, a failure in the Web server service).

Due to the importance of the advisor, when you start Manager, the Load Balancer GUI automatically displays a pop-up window prompting you to start an advisor.

The Load Balancer product offers advisors for specific protocols and services, and a generic advisor called *Connect*.

In our scenario, we are load balancing a Web server using the HTTP protocol. Therefore, we use the default values as shown in Figure 5-20, which are HTTP in the Advisor name field and port 80 in the Port number field. These are the default values presented to us because we previously added port 80 in our configuration.

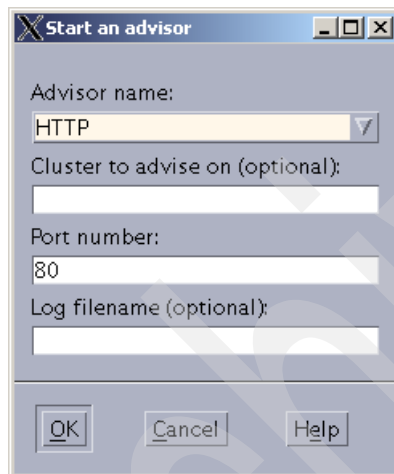


Figure 5-20 Starting the HTTP advisor

You can also choose a specific cluster with which to associate this advisor. By leaving the optional *Cluster to advise on* field blank, this advisor is automatically associated with all clusters that are load balancing port 80.

If you want to specify a log filename for this advisor, type in the desired name in the Log filename field. The default filename for the HTTP advisor is `Http_80.log`.

We have concluded the basic load balancing configuration, so the last thing we need to do is to save the configuration performed so far.

12. Right-click **Host: lb.itso.ibm.com** and select **Save Configuration File As...**, as shown in Figure 5-21 on page 147.

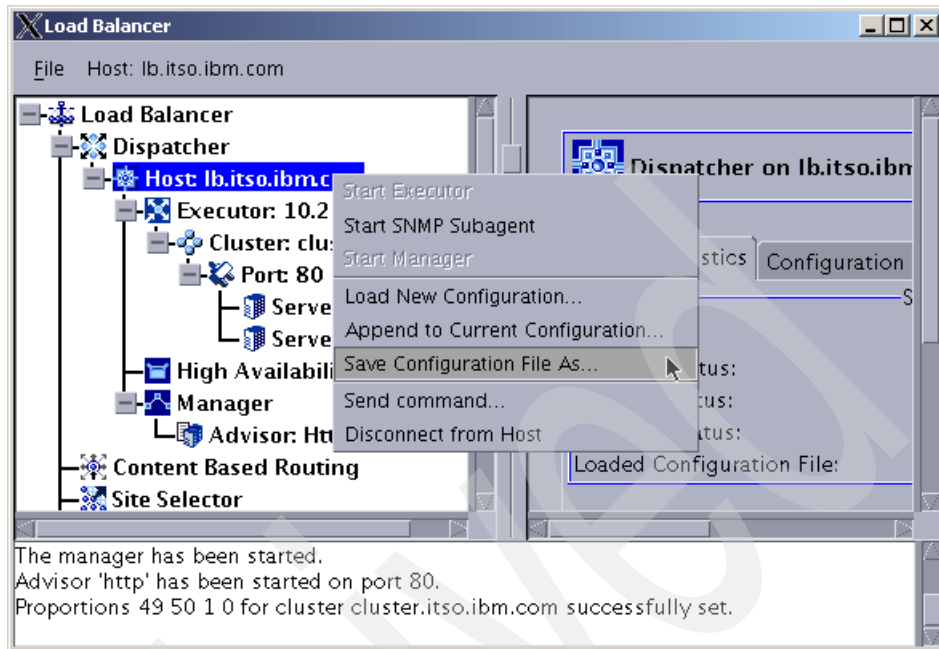


Figure 5-21 Saving the Load Balancer configuration

A pop-up window is displayed. In the Filename field, you can either select an existing configuration file (which will be overwritten) or you can enter a new filename.

The filename default.cfg is the default name for Load Balancer. This means that when you start the Dispatcher server (**dsserver**), it will look for the file default.cfg and, if it exists, it will load it. default.cfg is stored in <LB_install_path>/servers/configurations/dispatcher.

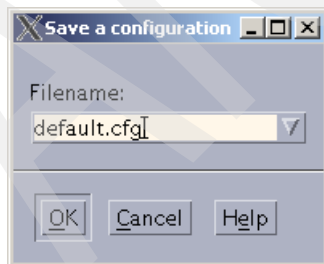


Figure 5-22 Choosing the configuration filename

The resulting configuration file is shown in Example 5-3 on page 148. Please note that each individual command has to be on one line in the configuration

file. However, because of size limitations, some lines might be printed on two lines in our examples.

Example 5-3 Configuration file for the basic scenario

```
dscontrol set loglevel 1
dscontrol executor start

dscontrol cluster add cluster.itso.ibm.com address 10.20.10.100 primaryhost
10.20.10.102

dscontrol cluster set cluster.itso.ibm.com proportions 49 50 1 0
dscontrol executor configure 10.20.10.100 en0 255.255.255.0

dscontrol port add cluster.itso.ibm.com:80 reset no

dscontrol server add cluster.itso.ibm.com:80:http2 address 10.20.10.104
dscontrol server add cluster.itso.ibm.com:80:http1 address 10.20.10.103

dscontrol manager start manager.log 10004

dscontrol advisor start Http 80 Http_80.log
```

If you do not want to use the Load Balancer GUI to configure the scenario described here, you can copy the commands shown in Example 5-3 into your own default.cfg file, and when you run **dsserver**, it will automatically be loaded.

You can also type those commands into the operating system prompt, one by one.

Note that in either case, you need to change the hostnames and IP addresses shown here to the appropriate ones for your environment.

5.2.2 Configuring the balanced servers

Although Dispatcher is ready to begin load balancing the traffic, there is still one more configuration step to perform: preparing the balanced servers to accept packets sent to the cluster IP address.

We do this by adding an IP alias to the loopback interface of these servers. This alias is the IP address of the cluster.

Important: The cluster IP address must be added to the balanced servers as a non-advertising address (the server must not respond to ARP requests for that address). That is why we use it as an IP alias to the loopback interface.

We describe how to add this IP alias for Windows 2000 and AIX machines in “Aliasing the loopback interface in Windows 2000 systems” on page 149 and “Aliasing the loopback interface in AIX systems” on page 155.

Aliasing the loopback interface in Windows 2000 systems

This procedure may add a new route to your routing table, so we recommend you save an output of your current routing table to use later. See Example 5-4.

Example 5-4 Original routing table

```
C:\> route print
=====
Interface List
0x1 ..... MS TCP Loopback interface
0x1000003 ...00 02 55 91 4b 4c ..... AMD PCNET Family Ethernet Adapter
=====

Active Routes:
Network Destination        Netmask          Gateway          Interface        Metric
0.0.0.0                    0.0.0.0          10.20.10.1       10.20.10.103     1
10.20.10.0                 255.255.255.0    10.20.10.103     10.20.10.103     1
10.20.10.103              255.255.255.255  127.0.0.1        127.0.0.1        1
9.255.255.255             255.255.255.255  10.20.10.103     10.20.10.103     1
127.0.0.0                 255.0.0.0        127.0.0.1        127.0.0.1        1
224.0.0.0                 224.0.0.0        10.20.10.103     10.20.10.103     1
255.255.255.255          255.255.255.255  10.20.10.103     10.20.10.103     1
Default Gateway:          10.20.10.1
=====

Persistent Routes:
None
```

Now you need to install the MS Loopback Adapter (if you have not already done so).

1. Click **Start -> Settings -> Control Panel -> Add/Remove Hardware**. In the Add/Remove Hardware Wizard, click **Next**, select **Add/Troubleshoot a device**, and click **Next**.
2. Select **Add a new device** as shown in Figure 5-23 on page 150 and click **Next**.

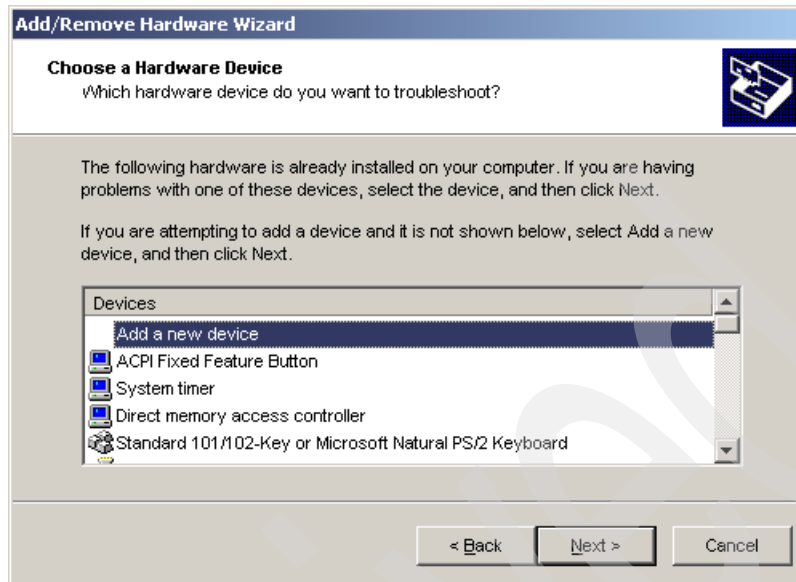


Figure 5-23 Adding a new device in Windows

3. In the Find New Hardware window select **No, I want to select the hardware from a list** as shown in Figure 5-24 and click **Next**.



Figure 5-24 Find new hardware window

4. In the Hardware Type window, select **Network Adapters** as shown in Figure 5-25 on page 151 and click **Next**.

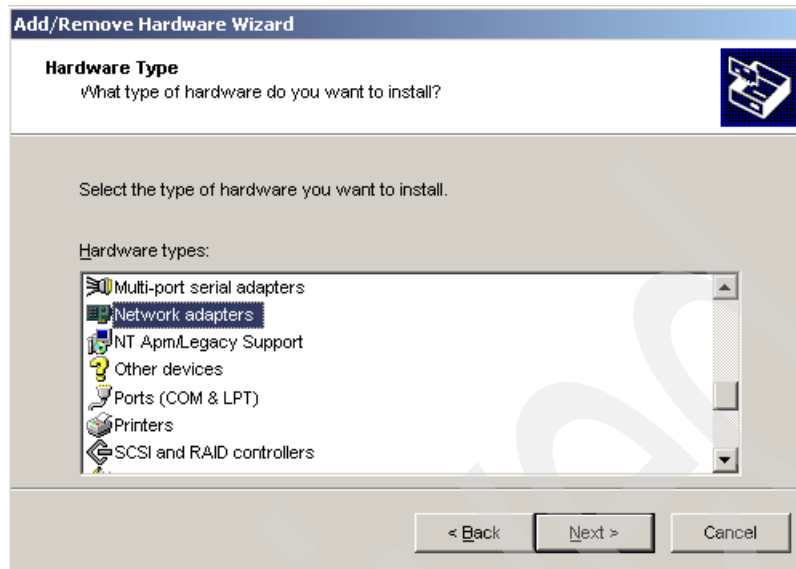


Figure 5-25 Adding a network adapter

5. In the Select Network Adapter window, locate and select **Microsoft** from the list on the left hand side, and select **Microsoft Loopback Adapter** from the list on the right hand side, as shown in Figure 5-26, then click **Next**.

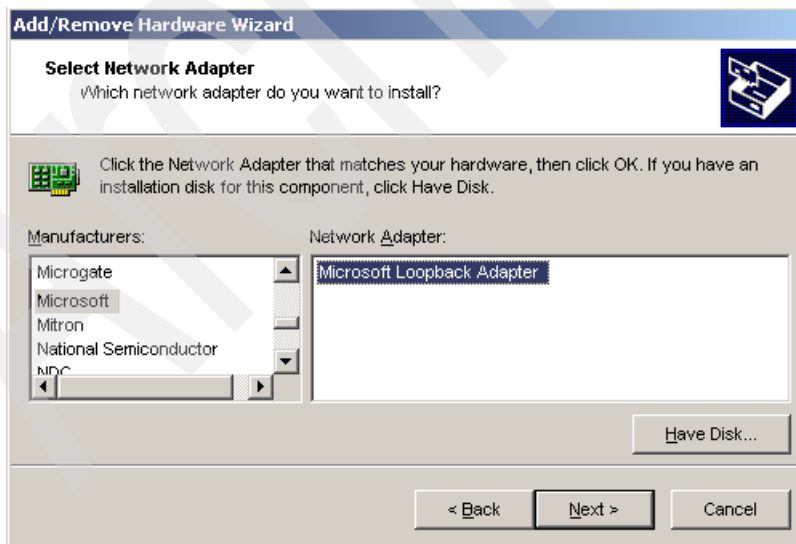


Figure 5-26 Adding the Microsoft Loopback Adapter

6. Confirm the selection in the following window by clicking **Next**. When the installation has finished, click the **Finish** button.
7. Now we need to configure this new adapter. Click **Start -> Settings -> Control Panel -> Network and Dial-up Connections** and locate the newly installed device. You can identify it by looking at the Device Name column, and locating the Microsoft Loopback Adapter device, as shown in Figure 5-27.

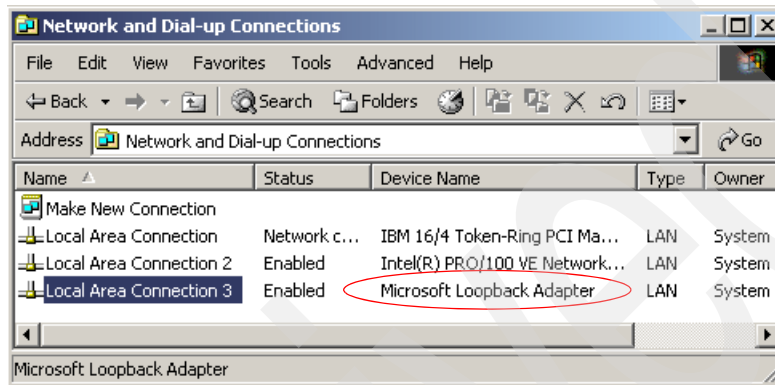


Figure 5-27 Locating the Microsoft Loopback Adapter device

8. Right-click the connection that represents the Microsoft Loopback Adapter and select **Properties**.
9. In the components list pane, click **Internet Protocol (TCP/IP)** and click the **Properties** button, as shown in Figure 5-28.

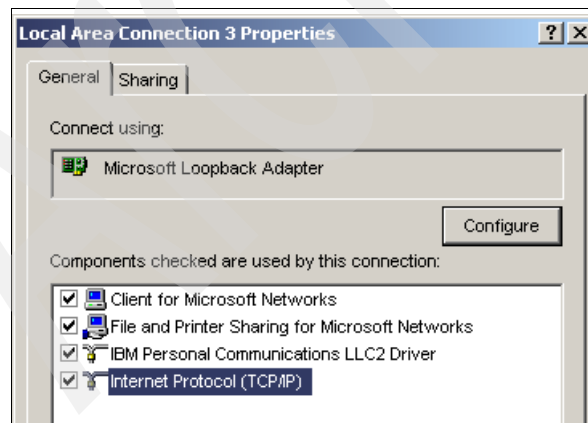


Figure 5-28 Local Area Connection properties

10. In the Internet Protocol (TCP/IP) Properties window, click **Use the following IP address** and type the cluster IP address into the IP address field. Then enter the mask of your network in the *Subnet mask* field, as shown in Figure 5-29.

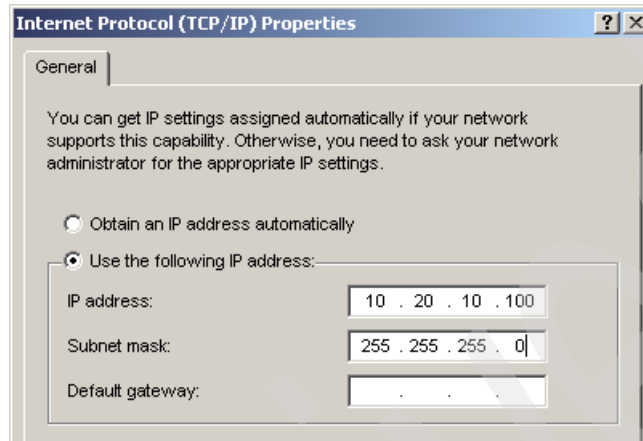


Figure 5-29 Configuring the cluster IP address in the loopback

11. Leave all other fields blank and click **OK**.
12. After the loopback device is enabled, check the routing table. Compare it to the one you saved in the beginning of this configuration (see the original routing table in Example 5-4 on page 149 and the new routing table in Example 5-5).

Example 5-5 Routing table after adding the loopback adapter

```
C:\> route print
```

```
=====
```

```
Interface List
```

```
0x1 ..... MS TCP Loopback interface
```

```
0x1000003 ...00 02 55 91 4b 4c ..... AMD PCNET Family Ethernet Adapter
```

```
0x3000004 ...02 00 4c 4f 4f 50 ..... MS LoopBack Driver
```

```
=====
```

```
Active Routes:
```

Network	Destination	Netmask	Gateway	Interface	Metric
	0.0.0.0	0.0.0.0	10.20.10.1	10.20.10.103	1
	10.20.10.0	255.255.255.0	10.20.10.100	10.20.10.100	1
	10.20.10.0	255.255.255.0	10.20.10.103	10.20.10.103	1
	10.20.10.100	255.255.255.255	127.0.0.1	127.0.0.1	1
	10.20.10.103	255.255.255.255	127.0.0.1	127.0.0.1	1
	10.255.255.255	255.255.255.255	10.20.10.100	10.20.10.100	1
	10.255.255.255	255.255.255.255	10.20.10.103	10.20.10.103	1

```

        127.0.0.0      255.0.0.0      127.0.0.1      127.0.0.1  1
        224.0.0.0      224.0.0.0      10.20.10.100   10.20.10.100 1
        224.0.0.0      224.0.0.0      10.20.10.103   10.20.10.103 1
        255.255.255.255 255.255.255.255 10.20.10.103   10.20.10.103 1
Default Gateway:      10.20.10.1
=====
Persistent Routes:
    None

```

Note that after the loopback adapter was added, the system also added three extra routes to the routing table. Now, there are three sets of routes to the same destination using two different gateways: first, the cluster IP address that was added to the loopback (10.20.10.100), and second the Ethernet adapter IP address (10.20.10.103).

From the three sets of repeated routes, the one that may cause routing problems is the one that was created for the local network, using the cluster IP address as the gateway:

```

10.20.10.0      255.255.255.0      10.20.10.100   10.20.10.100  1

```

13. The gateway is incorrect, and you need to remove this route. You can use the following command in a command prompt window:

```

C:\> route delete 10.20.10.0 10.20.10.100

```

14. This command must also be run after each reboot, because every time the loopback adapter is activated, the route is added back to the system. Therefore, we create a batch file, C:\routedel.bat, and add the following lines to it:

```

@echo off
route delete 10.20.10.0 10.20.10.100
exit

```

15. In order to run this batch file automatically after a reboot, we add it to the Registry. Run the command **regedit** and locate the key **HKEY_LOCAL_MACHINE -> SOFTWARE -> Microsoft -> Windows -> CurrentVersion -> Run**. On the menu bar, select **Edit -> New -> String Value**. Rename the new string value name to a name that makes sense to you, then double-click it so you can change the value data field. Enter C:\routedel.bat and click **OK**.

This batch file will be run after a reboot and it will delete that second route. If you need to add more aliases to the loopback, add the route delete for each alias to this same batch file.

Note: Due to a characteristic of the operating system, this batch file added to the Run registry entry will only run after a user logs in.

In order to have this batch file run after a reboot even if no user logs in, you need to create a Windows service for it. Refer to the operating system documentation for more information about how to create services.

Aliasing the loopback interface in AIX systems

The only thing you need to do in AIX in order to add the IP alias is to run the **ifconfig** command, as follows:

```
# ifconfig lo0 alias 10.20.10.100 netmask 255.255.255.255
```

Important: Make sure you use the `netmask` option with the `netmask 255.255.255.255` when you are adding an IP alias to the loopback device in AIX. If you use an incorrect netmask, or do not use this option at all, a new route will be added to your routing table using the loopback as the gateway, which causes routing problems.

Add this command to the end of the `/etc/rc.net` file so it will be run every time the networking configuration is run (for example, after a system reboot).

If you want to confirm that the alias was added to the loopback adapter, run the following command:

```
# ifconfig lo0
```

You should see an output similar to the one shown in Example 5-6.

Example 5-6 New IP alias added to the loopback device in AIX

```
# ifconfig lo0
lo0:
flags=e08084b<UP,BROADCAST,LOOPBACK,RUNNING,SIMPLEX,MULTICAST,GROUPRT,64BIT>
  inet 127.0.0.1 netmask 0xff000000 broadcast 127.255.255.255
  inet6 ::1/0
  inet 10.20.10.100 netmask 0xffffffff broadcast 10.20.10.100
  tcp_sendspace 65536 tcp_recvspace 65536
```

You can now test if your server is responding to requests sent to the cluster IP address by using a browser on the local machine and requesting the IP address of the cluster.

Other examples of aliasing the loopback interface

The following table is a list of commands to add an alias on different platforms:

Table 5-1 Adding an IP alias

Platform	Command
AIX 4.3	ifconfig lo0 alias <cluster_IP_address> netmask <netmask>
AIX 5.x	ifconfig lo0 alias <cluster_IP_address> netmask 255.255.255.255
HP-UX	ifconfig lo0:1 <cluster_IP_address> up
Linux	ifconfig lo:1 <cluster_IP_address> netmask 255.255.255.255 up For the second alias, use lo:2, and so forth.
OS/2	ifconfig lo <cluster_IP_address>
Solaris 7	ifconfig lo0:1 <cluster_IP_address> 127.0.0.1 up
Solaris 8 and 9	ifconfig lo0:1 plumb <cluster_IP_address> netmask <netmask> up

Restriction: Some operating systems do not allow adding an alias to the loopback adapter. One solution is to change the loopback IP address, but you will not be able to use this server on more than one cluster.

After adding the alias, check for the extra route that may have been created, and remove it according to the correct procedure for each operating system.

Note: If you test the access with a browser and do not get a response, but other services are responding to the cluster IP address (FTP, for example), you may need to configure your HTTP server to listen to the IP address of the cluster (refer to the documentation of your HTTP server for more information about how to do this).

5.2.3 Testing the basic scenario

Dispatcher provides reports that can be used to verify the configuration. You can see whether the back-end servers that make up the cluster are active and sending responses to the advisors. You can also see if the traffic is being balanced using the server monitor on the GUI.

Example 5-7 on page 157 shows the output of the **dscontrol manager report** command. The first table lists the back-end servers being load balanced and their status. The second table lists the servers by port, weight, number of active and new connections, and load values.

The last table shows the advisors that were started, the port, and the timeout value attributed to it.

Example 5-7 Manager report example

SERVER				IP ADDRESS		STATUS	
http2				10.20.10.104		ACTIVE	
http1				10.20.10.103		ACTIVE	
MANAGER REPORT LEGEND							
ACTV	Active Connections						
NEWC	New Connections						
SYS	System Metric						
NOW	Current Weight						
NEW	New Weight						
WT	Weight						
CONN	Connections						
cluster.itso.ibm.com							
10.20.10.100		WEIGHT		ACTV	NEWC	PORT	SYS
PORT: 80		NOW	NEW	49%	50%	1%	0%
http2		10	10	1	9	28	0
http1		9	9	1	9	34	0
ADVISOR		CLUSTER:PORT			TIMEOUT		
http		80			unlimited		

You can check whether packets are being forwarded to the cluster by issuing the **dscontrol executor report** command, which produces a report of the packet traffic on the Executor component of Dispatcher, as shown in Example 5-8.

Example 5-8 Executor report example

Executor Report:

```
-----
Version level ..... 06.00.00.00
Total packets received since starting ..... 591,860
Packets sent to nonforwarding address ..... 1,300
Packets processed locally on this machine ..... 0
Packets sent to collocated server ..... 0
Packets forwarded to any cluster ..... 200,251
Packets not addressed to active cluster/port .. 20,631
```

```
KBytes transferred per second ..... 3,259
Connections per second ..... 97
Packets discarded - headers too short ..... 0
Packets discarded - no port or servers ..... 0
Packets discarded - network adapter failure ... 0
Packets with forwarding errors..... 0
```

We can use the server monitor on the GUI to graphically view the load being distributed among the servers. It is available per port and per server. If you right-click a desired port or a specific server, you can select the **Monitor...** option.

The Monitor tool provides the same information that you can view in the Manager report, but it dynamically updates the data and shows it in a chart.

If you choose to monitor a port, there will be one bar or line for each server configured on that port. If you choose to monitor a specific server then there will be only one bar or line for the chosen server.

By default, the chart presents the Weight of the servers (see Figure 5-30). You can also select Port load (the load value provided by the Advisor), System load (the load value provided by the Metric Server), Active connections and New connections.

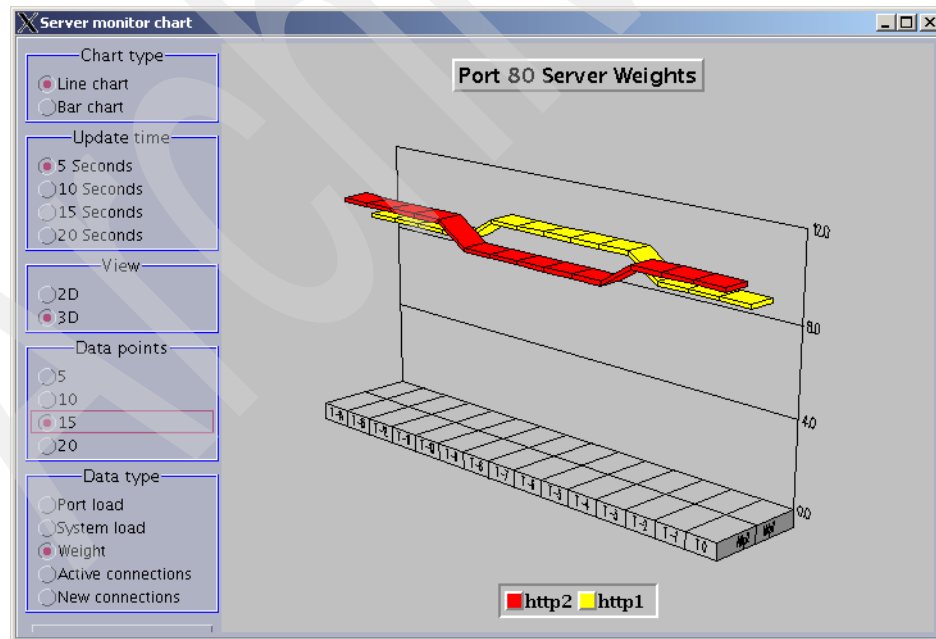


Figure 5-30 Monitor tool

In order to monitor the behavior of the cluster, you can try repeatedly selecting a clustered page using the browser, or you can use an HTTP benchmarking tool to send requests to the cluster. In our lab, we used Rational® Performance Tester to issue a burst of requests to the home page of our cluster (refer to 17.3.3, “Rational Performance Tester” on page 956 for more information about this tool).

Using the Monitor tool, we can see the distribution of the load by selecting **New Connections** in the Data Type box, as shown in Figure 5-31.

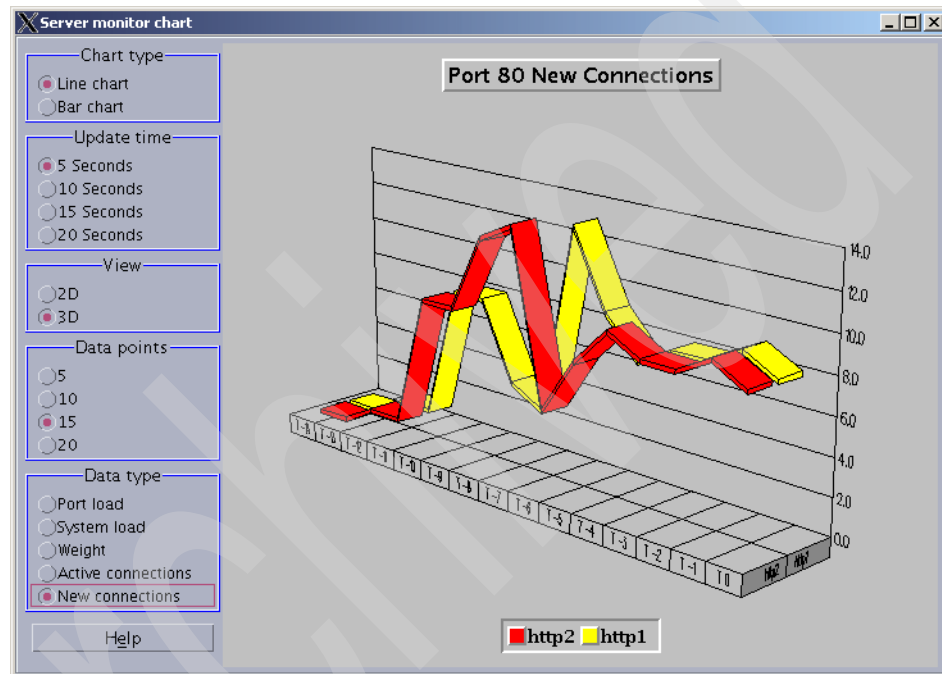


Figure 5-31 Load distribution among two Web servers

In order to simulate a server failure, we stop the Web server process in server http2, and the Monitor chart shows that Dispatcher identifies that http2 is no longer available (the server is still up, but the process is no longer responding to the requests), and it stops sending requests to it. As shown in Figure 5-32 on page 160, all requests are then forwarded to the remaining Web server, http1.

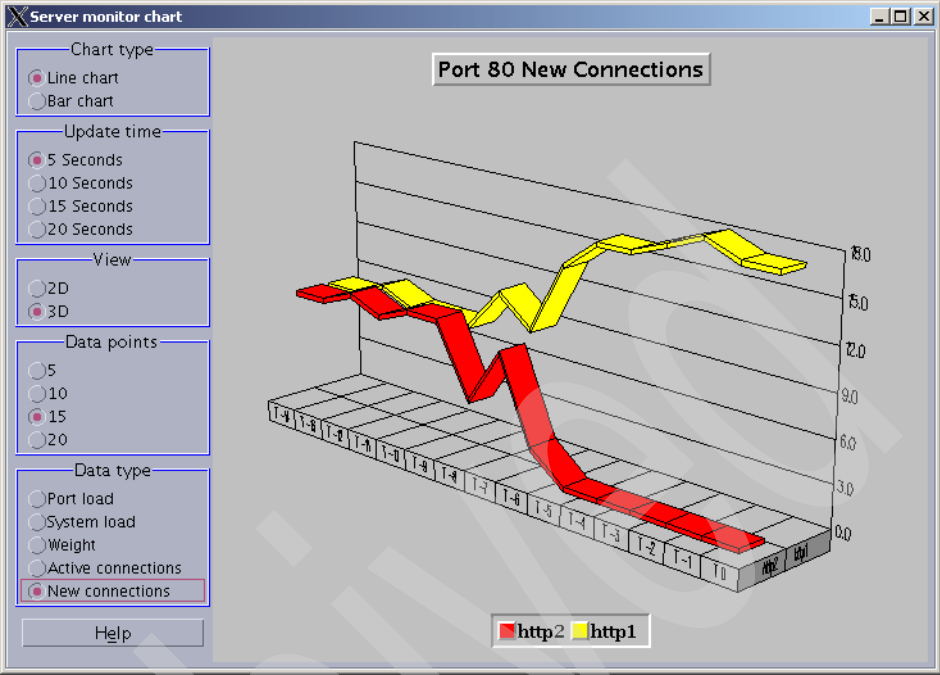


Figure 5-32 Dispatcher detects a failure in http2

This information is also available in the **dscontrol1 manager** report output, as shown in Example 5-9.

Example 5-9 Manager report - failure in server http2

SERVER		IP ADDRESS	STATUS
http2		10.20.10.104	ACTIVE
http1		10.20.10.103	ACTIVE

MANAGER REPORT LEGEND	
ACTV	Active Connections
NEWC	New Connections
SYS	System Metric
NOW	Current Weight
NEW	New Weight
WT	Weight
CONN	Connections

cluster.itso.ibm.com							
10.20.10.100	WEIGHT		ACTV	NEWC	PORT	SYS	
PORT: 80	NOW	NEW	49%	50%	1%	0%	
http2	0	0	1	0	-1	0	
http1	11	11	1	17	35	0	

ADVISOR	CLUSTER:PORT	TIMEOUT
http	80	unlimited

Note that the PORT column shows the value -1 for the http2 server. This means that the advisor is getting no response from this server. That makes the weight of this server set to zero (see column WEIGHT).

After a while, we restarted the Web server process in server http2. The Server monitor chart in Figure 5-33 shows that Dispatcher identifies that the server is available again, and distributes part of the load to it. The line for the http1 server starts dropping because its load starts being shared with http2.

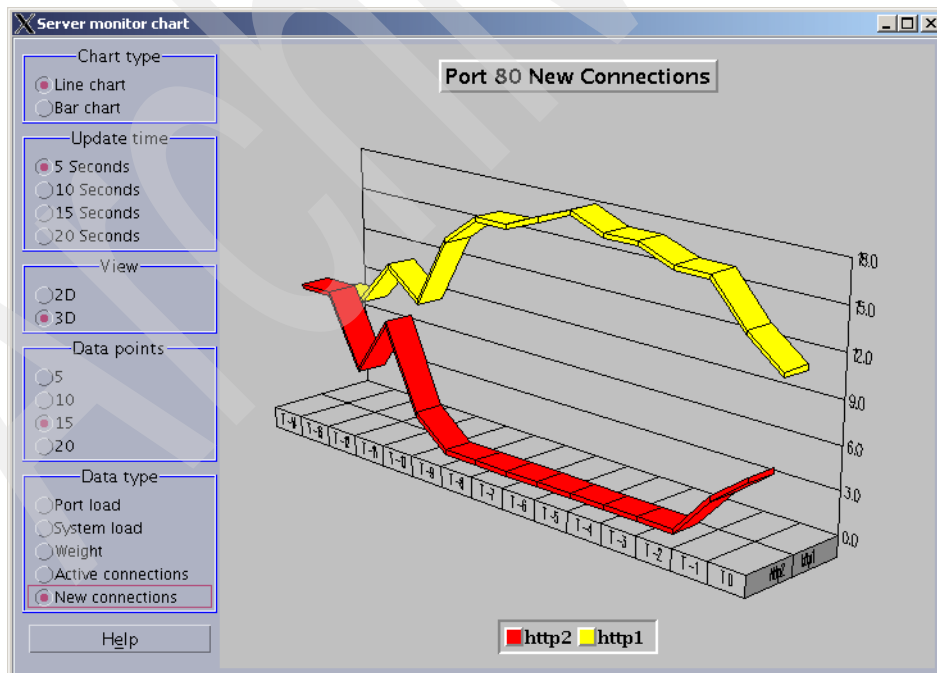


Figure 5-33 Dispatcher detects that http2 is back up

5.3 Load Balancer: high availability scenario

This scenario extends the configuration shown in 5.2, “Load Balancer configuration: basic scenario” on page 135 by adding a backup Load Balancer server and a high availability configuration.

We recommend that you first set up the basic scenario and test the load balancing. Once this is working, follow the instructions below to add the second Load Balancer server and the high availability configuration.

5.3.1 Configuring high availability

You can use the configuration file that was created for the basic scenario. We assume that the basic scenario is already set up and working.

When you are using the high availability feature of Load Balancer, you do not use the **dscontrol cluster configure** command in the Dispatcher configuration file. For a high availability configuration, we create scripts that control all IP aliases that need to be added or removed from the network interfaces and the loopback interface, depending on the state of the server.

Leaving the **dscontrol cluster configure** command in the configuration would break the high availability configuration because it adds the IP alias to the network interface no matter what the state of the Load Balancer server is (active or standby). We need to make sure that the IP alias is only added to the network interface when a server changes to the active state. For more information, refer to 5.3.4, “Configuring the high availability scripts” on page 171.

First, we have to remove the cluster IP alias from the existing basic configuration before proceeding.

1. Open the Load Balancer GUI and connect to the primary server as described in steps 1 on page 136 through 4 on page 137. Make sure that the basic configuration is loaded by checking that the cluster, port, and servers are already configured.
2. Right-click **Executor: 10.20.10.102** and select **Unconfigure Interface Address...**, as shown in Figure 5-34 on page 163.

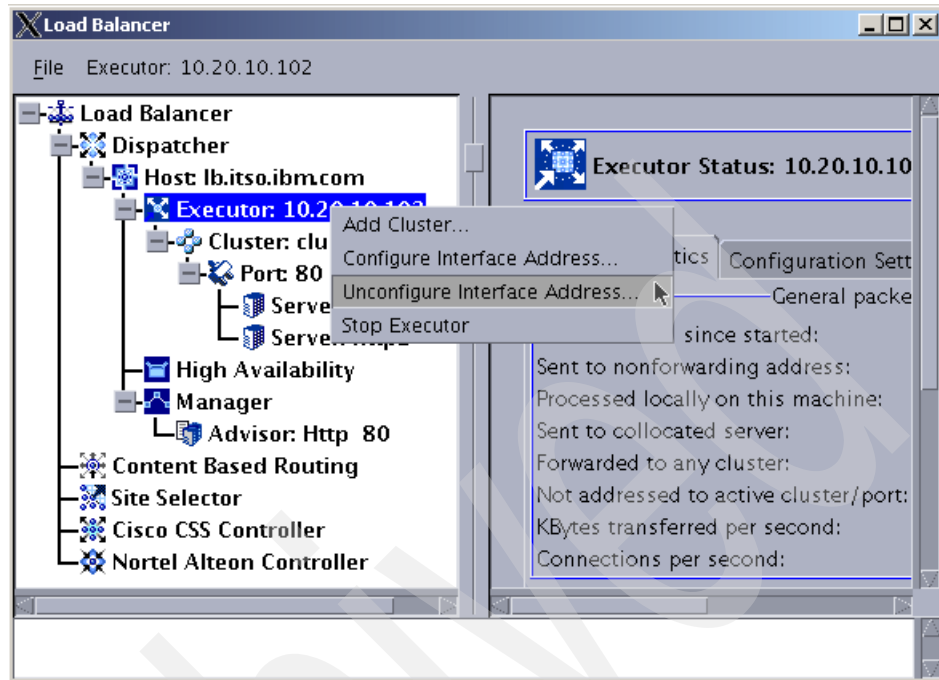


Figure 5-34 Unconfiguring a cluster interface address

A pop-up window is displayed; enter the IP address of the cluster in the Interface address field. In our scenario, we want to remove the IP alias for the cluster address 10.20.10.100, as shown in Figure 5-35.

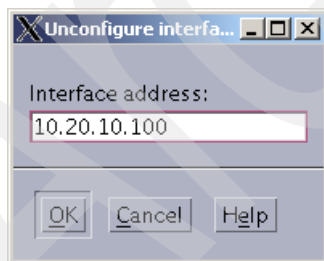


Figure 5-35 Unconfiguring the cluster address 10.20.10.100

3. Save the current configuration then copy it to the backup Dispatcher server, so you do not need to set up the basic load balancing configuration there again.

Tip: To save the configuration file, right-click **Host** -> **Save Configuration File As....**

4. Now we need to add the high availability configuration. Right-click **High Availability** in the left pane of the GUI window and select **Add High Availability Backup....**, as shown in Figure 5-36.

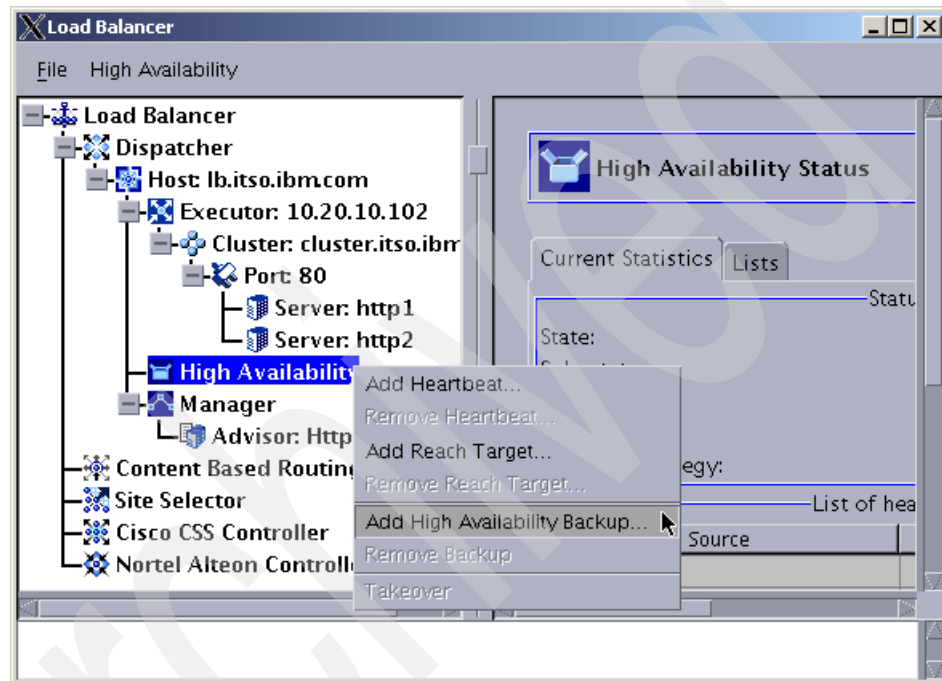


Figure 5-36 Adding a high availability backup entry

A new pop-up window is displayed as shown in Figure 5-37 on page 165.

First, you need to choose a port number that will be used by both Dispatcher servers to exchange the information needed to keep them synchronized, and fill it into the Port number field. You can choose any port, you just need to make sure that the port number matches on both servers. For our scenario, we chose port **12345**.

In the Role field, select the role that this machine will have in the high availability scenario (Primary, Backup, or Both). In our scenario, this machine is our primary machine, so we selected **Primary**.

In the Recovery strategy field, choose how your primary machine is going to behave in case the backup machine has taken over. If you select **Auto**, it will

take over as soon as it is up (and responding to the network). If you select **Manual**, you will need to take an action (either using the Load Balancer GUI or running the command `dscontrol high takeover`) to force it to take over. As long as you do not force the takeover, the backup remains active. In our scenario, we selected **Auto**.

The last thing you need to add in this window is the heartbeat source address and the heartbeat destination address. The heartbeat is a GRE (Generic Route Encapsulation) packet that is sent from the local machine to the other server in the same high availability cluster to make sure it is responding. Enter the IP address of the local machine into the Heartbeat source address field, and the IP address of the backup machine into the Heartbeat destination address field.

When you are finished, click **OK**.

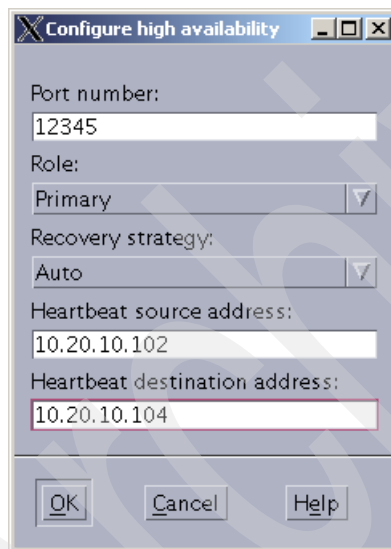


Figure 5-37 Configuring high availability for the primary server

For now, the configuration is done on the primary Dispatcher server. Click **High Availability** in the left pane, and you can see the high availability status information in the right pane, as shown in Figure 5-38 on page 166.

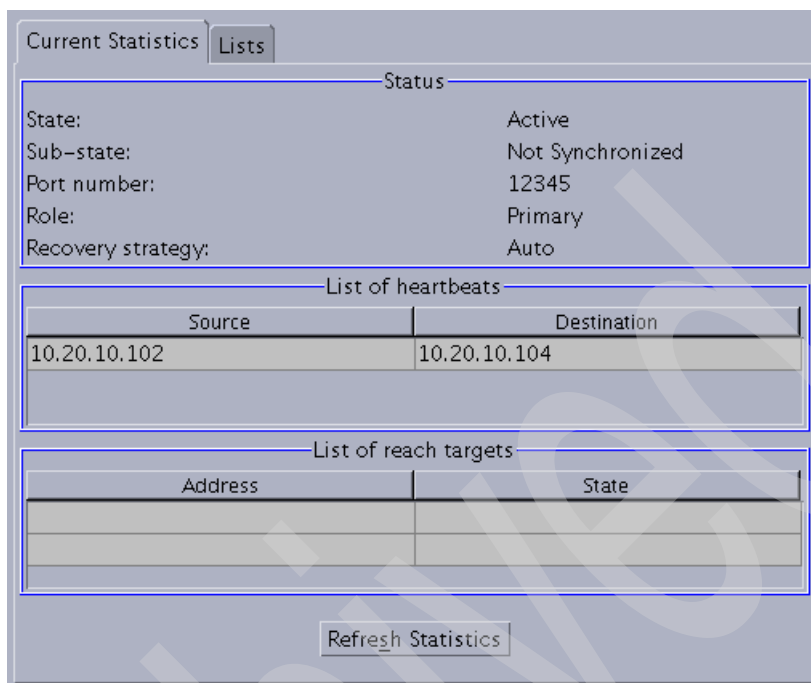


Figure 5-38 High availability status information

The next steps must be performed on the backup Dispatcher server, which in our scenario is the server http2.

1. Open the Load Balancer GUI and connect to the backup server as described in steps 1 on page 136 through 4 on page 137. In our scenario, the backup server is http2.itso.ibm.com, as shown in Figure 5-39; the backup server is collocated with the Web server.

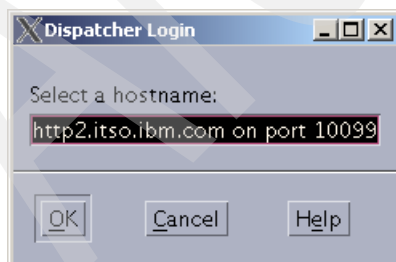


Figure 5-39 Connecting to the backup Load Balancer server

2. First, you have to load the configuration file you copied from the primary server in step 3 on page 163. Right-click **Host: http2.itso.ibm.com** and select **Load New Configuration...** as shown in Figure 5-40.

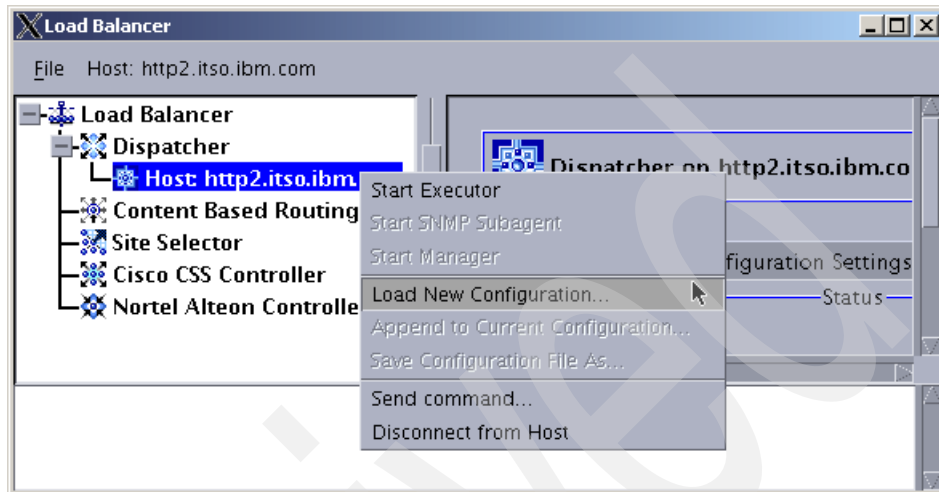


Figure 5-40 Loading a configuration

Select the filename in the pop-up window and click **OK**. See Figure 5-41.

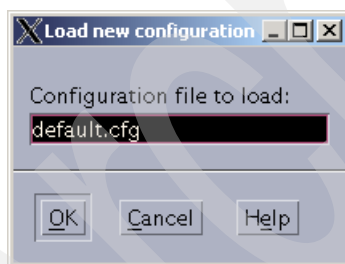


Figure 5-41 Loading the default configuration file

3. As mentioned earlier, we are using a collocated scenario for the backup Dispatcher server (which means that it is running on the same system as one of the Web servers). This requires a change in the configuration to indicate that http2 is a collocated server. In our case, this change is only required on the backup server, and you do not need to do it if you use a dedicated server.

Click the collocated server name in the left pane under Port:80 (in our scenario, we selected **http2**). Locate the Collocated field in the right pane, select the option **yes** and click **Update Configuration** at the bottom of this pane, as shown in Figure 5-42 on page 168.

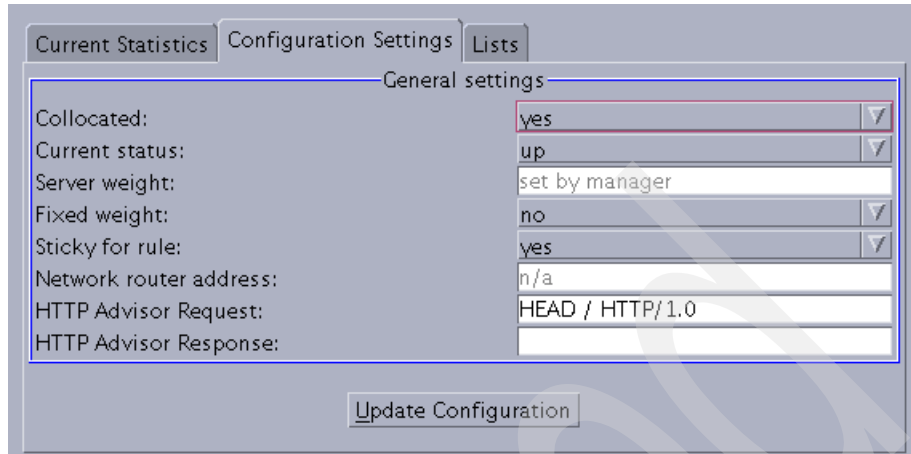


Figure 5-42 Configuring the collocated option

4. Add the high availability information for the backup server. Right-click **High Availability** in the left pane of the GUI window and select **Add High Availability Backup...** (this is the same procedure we performed for the primary server in step 4 on page 164).

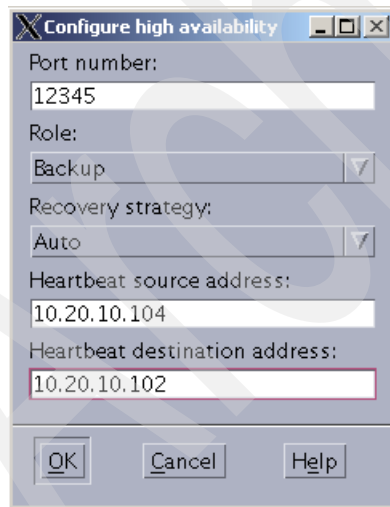


Figure 5-43 Configuring high availability in the backup server

You need to use the same parameters for Port number and Recovery strategy that you used in the configuration of the primary server (see Figure 5-37 on page 165).

Select **Backup** in the Role field.

For the backup server, the Heartbeat source address is the backup server itself, and the Heartbeat destination address is the primary server, as shown in Figure 5-43 on page 168.

5. Save the configuration of the primary and backup servers.

Refer to 5.3.3, “Checking the configuration” on page 170 for the complete configuration files of both Load Balancer servers.

5.3.2 Adding reach targets

The final step in the configuration of our high availability scenario is to add the reach target.

A reach target works in a similar way to the heartbeat, but this time we use another machine as the destination for the ping packet. The same reach targets are added to the configuration of both primary and backup servers.

If the active Load Balancer cannot reach this target (it does not receive a response from the ping packet), but the standby server still receives responses from it, the standby server will force a failover. Therefore, it is very important that you choose a stable server or network appliance as the reach target. We usually recommend using the default router of the local network (use the IP of the interface that is directly connected to the local network).

If you configure more than one reach target, the standby Load Balancer will fail over if it receives responses from more reach targets than the active Load Balancer.

In our scenario, we used the IP address of our local router, which is 10.20.10.1.

To add this IP address as a reach target IP address, right-click **High Availability** in the left pane of the window and select **Add Reach Target...** A pop-up window is displayed, as shown in Figure 5-44.

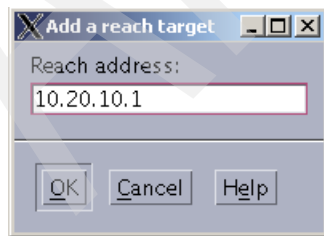


Figure 5-44 Adding a reach target

Type the IP address of the destination you chose, and click **OK**. Make sure you add the same reach target in both the primary and the backup Dispatcher servers' configurations. Save the configuration file.

You can add more than one reach target if you need to.

Note: We recommend that you add the reach target after you have already tested the high availability configuration (including takeover and failover) and the load balancing.

The reason for this is that you could experience unwanted failovers during your initial high availability tests if the reach target system is unstable.

5.3.3 Checking the configuration

The resulting configuration file for our primary server is shown in Example 5-10. Please note that each individual command has to be on one line in the configuration file. However, because of size limitations, some lines might be printed on two lines in our examples.

Example 5-10 Primary server configuration file

```
dscontrol set loglevel 1
dscontrol executor start

dscontrol highavailability heartbeat add 10.20.10.102 10.20.10.104
dscontrol highavailability backup add primary=10.20.10.102 auto 12345
dscontrol highavailability reach add 10.20.10.1

dscontrol cluster add cluster.itso.ibm.com address 10.20.10.100 primaryhost
10.20.10.102
dscontrol cluster set cluster.itso.ibm.com proportions 49 50 1 0

dscontrol port add cluster.itso.ibm.com:80 reset no

dscontrol server add cluster.itso.ibm.com:80:http2 address 10.20.10.104
dscontrol server add cluster.itso.ibm.com:80:http1 address 10.20.10.103

dscontrol manager start manager.log 10004

dscontrol advisor start Http 80 Http_80.log
```

The configuration file of our backup server is shown in Example 5-11 on page 171.

Example 5-11 Backup server configuration file

```
dscontrol set loglevel 1
dscontrol executor start

dscontrol highavailability heartbeat add 10.20.10.104 10.20.10.102
dscontrol highavailability backup add backup auto 12345
dscontrol highavailability reach add 10.20.10.1

dscontrol cluster add cluster.itso.ibm.com address 10.20.10.100 primaryhost
10.20.10.102
dscontrol cluster set cluster.itso.ibm.com proportions 49 50 1 0

dscontrol port add cluster.itso.ibm.com:80 reset no

dscontrol server add cluster.itso.ibm.com:80:http2 address 10.20.10.104
dscontrol server set cluster.itso.ibm.com:80:http2 collocated y
dscontrol server add cluster.itso.ibm.com:80:http1 address 10.20.10.103

dscontrol manager start manager.log 10004

dscontrol advisor start Http 80 Http_80.log
```

5.3.4 Configuring the high availability scripts

After configuring high availability on both Dispatcher servers, you now need to create the high availability scripts (refer to 4.6, “Dispatcher scripts” on page 119).

You need to configure at least the goActive, goStandby, and goInOp scripts but we also show how to use the serverUp, serverDown and highavailChange scripts.

You can start working with the samples available in the <product_install_path>/servers/samples directory and customize them for your environment, or you can write them from scratch.

All scripts are created in the <LB_install_path>/servers/bin directory, and you need to name them exactly as indicated (Load Balancer is case sensitive).

Note: The scripts are identical for the primary and the backup Dispatcher servers, unless there is some particular command you need to run on each machine. This might be the case, for example, if your backup Dispatcher is collocated while your primary Dispatcher is not. In this case, you need to delete/add the loopback alias commands on the collocated server.

We used two AIX V5.2 servers so we wrote the scripts using ksh syntax.

goActive

Executor runs this script whenever the Load Balancer server switches to active state. For example, when a fail over occurs and the standby server switches to active, or when the Load Balancer is started in the primary server and the recovery strategy is set to automatic.

This script is configured to remove the cluster IP alias (10.20.10.100) from the loopback (lo0) and add it to the network interface (defined in the INTERFACE variable). As mentioned earlier, we removed the **dscontrol executor configure** command from the configuration in step 2 on page 162 because when we implement high availability we do all IP aliasing through the scripts. See Example 5-12.

We added a text message that will be written to a log file using the **echo** command. Note that you can also start or stop processes, you can use **while/for/if** and other ksh commands.

It is important that each individual command be on one line in the configuration file.

Example 5-12 goActive script

```
#!/bin/ksh
ND_LOGDIR=/opt/ibm/edge/lb/servers/logs/dispatcher
CLUSTER1=10.20.10.100
INTERFACE=en0
NETMASK=255.255.255.0

date >> $ND_LOGDIR/ha.log
echo "This machine is Active. Aliasing cluster address(es) to NIC" >>
$ND_LOGDIR/ha.log

ifconfig lo0 delete $CLUSTER1
ifconfig $INTERFACE alias $CLUSTER1 netmask $NETMASK
```

Note: Only delete the loopback alias (in our example, we do it with the line “ifconfig lo0 delete \$CLUSTER1”) if you are doing both MAC forwarding and collocation on the Load Balancer machine.

goStandby

Executor runs this script whenever the Load Balancer server switches to standby state. For example, when the Load Balancer is started in the backup server and the primary server is active.

This script is configured to remove the cluster IP alias (10.20.10.100) from the network interface (defined in the INTERFACE variable) and add it to the loopback (lo0). See Example 5-13.

Example 5-13 goStandby script

```
#!/bin/ksh
ND_LOGDIR=/opt/ibm/edge/lb/servers/logs/dispatcher
CLUSTER1=10.20.10.100
INTERFACE=en0
NETMASK=255.255.255.0

date >> $ND_LOGDIR/ha.log
echo "Deleting the device aliases and adding the loopback aliases" >>
$ND_LOGDIR/ha.log

ifconfig en0 delete $CLUSTER1
ifconfig lo0 alias $CLUSTER1 netmask 255.255.255.255
```

Note that when we configure an IP alias to the loopback we use the full netmask (255.255.255.255) in order to prevent the operating system from adding extra routes to the routing table.

Note: Only add the loopback alias (in our example, we do it with the line “ifconfig lo0 alias \$CLUSTER1 netmask 255.255.255.255”) if you are doing both MAC forwarding and collocation on the Load Balancer machine.

goInOp

Executor runs this script whenever Executor is stopped or it is first started, so we want to remove the IP aliases from all network interfaces and loopback. See Example 5-14.

Example 5-14 goInOp script

```
#!/bin/ksh
ND_LOGDIR=/opt/ibm/edge/lb/servers/logs/dispatcher
CLUSTER1=10.20.10.100
INTERFACE=en0
NETMASK=255.255.255.0

date >> $ND_LOGDIR/ha.log
echo "Executor has stopped. Removing loopback and device aliases. \n" >>
$ND_LOGDIR/ha.log

ifconfig lo0 delete $CLUSTER1
ifconfig $INTERFACE delete $CLUSTER1
```

Note: Only delete the loopback alias (in our example, we do it with the line “ifconfig lo0 delete \$CLUSTER1”) if you are doing both MAC forwarding and collocation on the Load Balancer machine.

serverDown

Executor runs this script whenever a balanced server is marked down by Manager.

We use this script to record an entry in a log file informing that one of the HTTP servers was marked down by Manager. See Example 5-15.

Example 5-15 serverDown script

```
#!/bin/ksh
DATE=`date`
OUTPUT="$DATE $1 has been marked down."

echo $OUTPUT >> /opt/ibm/edge/lb/servers/logs/dispatcher/lb.log
```

serverUp

Executor runs this script whenever a balanced server is marked up by Manager.

We use this script to record an entry in a log file informing that one of the HTTP servers was marked up by Manager. See Example 5-16.

Example 5-16 serverUp script

```
#!/bin/ksh
DATE=`date`
OUTPUT="$DATE $1 has been marked back up."

echo $OUTPUT >> /opt/ibm/edge/lb/servers/logs/dispatcher/lb.log
```

Note: When Dispatcher runs the serverUp and serverDown scripts, it passes the name of the balanced server as the first parameter to the script. So the variable \$1 contains the server name in the format <cluster>:<port>:<server>.

For example, in our scenario the http1 server would be identified as cluster.itso.ibm.com:80:http1.

highavailChange

Executor runs this script whenever the state of the Load Balancer server changes (from active to standby or from standby to active).

We use this script to record an entry in a log file informing that the state of the local Load Balancer server has changed. See Example 5-17.

Example 5-17 highavailChange script

```
#!/bin/ksh
DATE=`date`
OUTPUT="$DATE LB just ran $1."

echo $OUTPUT >> /opt/ibm/edge/lb/servers/logs/dispatcher/lb.log
#echo $OUTPUT | mail -s "highavailChange" root@localhost
```

Note: When Dispatcher runs the highavailChange script, Dispatcher passes the name of the high availability script which was run (goActive, goStandby, goInOp or goldle) as the first parameter to the script. You can use it by referencing the variable \$1.

After creating and editing the scripts in our AIX systems, we need to add execution permission to the files, or they will not be able to run.

For better security, we added read, write and execute permissions for the owner of the file (root) and we removed all other permissions by running the commands shown in Example 5-18.

Example 5-18 Setting the file permissions

```
# cd /opt/ibm/edge/lb/servers/bin
# chmod 700 go*
# chmod 700 server*
# chmod 700 highavailChange

# ls -l
total 2066
-rwx----- 1 root    system      1141 Oct 15 18:07 goActive
-rwx----- 1 root    system      1073 Oct 15 16:49 goInOp
-rwx----- 1 root    system      1147 Oct 15 16:49 goStandby
-rwx----- 1 root    system      1090 Oct 15 16:49 highavailChange
-rwx----- 1 root    system    1046318 Oct 11 11:23 ibmlb
-rwx----- 1 root    system       527 Oct 11 11:23 lbpd
-rwx----- 1 root    system       933 Oct 15 16:49 serverDown
-rwx----- 1 root    system       929 Oct 15 16:49 serverUp
```

Configuring the high availability scripts in Windows

The approach to configuring the high availability scripts in Windows is the same as in AIX: first, you locate the scripts in the \samples directory and change them according to the instructions in the sample file. The filenames are the same in

Windows, and they must end with .cmd. Then you copy them to the \bin directory. Example 5-19 shows the sample goActive file in Windows. We have bolded the lines that need to be changed or contain important information.

Example 5-19 Sample goActive script in Windows

```
@echo off
echo 5639-D57, 5630-A36, 5630-A37, 5724-D18, 5724-H88, 5724-H89 (C) COPYRIGHT
International Business Machines Corp. 1998, 2004
echo All Rights Reserved * Licensed Materials - Property of IBM
rem
rem goActive script
rem
rem Configure this script when using the high availability feature of
rem Load Balancer.
rem
rem This script is executed when Dispatcher goes into the
rem 'Active' state and begins routing packets.
rem
rem This script must be placed in Load Balancer's bin directory (by default
rem this is
rem C:\Program Files\ibm\edge\lb\servers\bin) and it must have the extension
rem .cmd in order
rem to be executable.
rem
rem Modify CLUSTER, INTERFACE and NETMASK to match your environment.
rem
rem tr0=first Token ring adapter, en0=first Ethernet adapter
rem
rem NETMASK must be the netmask of your LAN. It may be hexadecimal or
rem dotted-decimal notation.
rem
rem CLUSTER must be in dotted-decimal format if you are deleting the loopback
rem aliasing (see below)
rem
set CLUSTER=your.cluster.in.dotteddecimalformat
set INTERFACE=tr0
set NETMASK=255.255.248.0
rem
rem Deleting loopback alias(es): Only delete the loopback alias if you are
rem doing both MAC
rem forwarding and collocation on the Load Balancer machine. Use the
rem 'netsh interface dump' command to determine your loopback interface name.
rem
rem echo "Deleting loopback alias(es)
```

```
rem call netsh interface ip delete address "Local Area Connection 1" %CLUSTER%  
rem  
echo "Adding device alias(es)"  
call dscontrol e config %CLUSTER% %INTERFACE% %NETMASK%
```

Note: You need to remove the “rem” in front of the **call netsh interface** command line and add the correct name for the loopback adapter (for example lo0) if you are using both MAC forwarding and collocation. This makes sure the loopback alias is deleted (or added in the goStandby script respectively).

5.3.5 Testing the high availability scenario

After you finish configuring both Dispatcher servers and all the scripts, we recommend that you first test the load balancing in each Dispatcher server separately. Stop Executor on both servers, start one of them and test the load balancing. Then stop Executor on the first server, start it on the second one and do the tests again. Once you confirmed that each Dispatcher server is distributing the load correctly, start them both, first the primary server and then the backup server.

Important: When testing the high availability scenario in a Windows environment using the first version of WebSphere Edge Components V6 (available December 2004) we encountered problems with the stop executor command as well as when trying to stop Executor in the GUI (the system crashed). This problem has been fixed with APAR PK00545.

Please call IBM Service to get the fix associated with this APAR.

If you are using WebSphere Edge Components V6.0.0.2 or higher, then the fix is already included in the code.

As an alternative to stopping the Executor you can disable the network interface or unplug the network cable for your tests.

Tests with automatic recovery strategy

Open the Load Balancer GUI, connect to the primary server and click **High Availability** in the left pane. The high availability status information is displayed in the right pane, as shown in Figure 5-45 on page 178.

The screenshot shows a window titled 'Current Statistics' with a 'Lists' tab selected. The window is divided into three main sections: 'Status', 'List of heartbeats', and 'List of reach targets'. The 'Status' section displays the following information:

State:	Active
Sub-state:	Synchronized
Port number:	12345
Role:	Primary
Recovery strategy:	Auto

The 'List of heartbeats' section contains a table with two columns: 'Source' and 'Destination'.

Source	Destination
10.20.10.102	10.20.10.104

The 'List of reach targets' section contains a table with two columns: 'Address' and 'State'.

Address	State
10.20.10.1	reachable

At the bottom of the window, there is a 'Refresh Statistics' button.

Figure 5-45 High availability status

It shows that the current server is the active server (State field) and that both servers are synchronized (Sub-state field). It also shows the port number (which needs to be the same on both servers), the role of this server and the recovery strategy.

The List of heartbeats shows the heartbeat connections that were configured and the List of reach targets shows the status of the connectivity test to each IP address configured as a reach target.

Repeat the same procedure on the backup server and compare the values of these fields.

This information can also be obtained by running the command **dscontrol high status**, as shown in Example 5-20.

Example 5-20 High availability status on primary server

```
# dscontrol high status

High Availability Status:
-----
Role ..... Primary
Recovery strategy .... Auto
```

```
State ..... Active
Sub-state ..... Synchronized
Primary host ..... 10.20.10.102
Port ..... 12345
Preferred target ..... 10.20.10.104
```

Heartbeat Status:

```
Count ..... 1
Source/destination ... 10.20.10.102/10.20.10.104
```

Reachability Status:

```
Count ..... 1
Address ..... 10.20.10.1 reachable
```

Check the network interfaces configuration on both Dispatcher servers. The active server should have the cluster IP address aliased to the network interface, and the standby server should have it aliased to the loopback interface.

In order to test the high availability configuration we need to simulate a failure on the current active server. A simple way to do that is to disable the network interface.

If we were using the manual recovery strategy we would first issue a takeover command, and check how it is performed before actually simulating a problem.

In our scenario, lb is the primary machine, and http2 is the backup machine. In order to simulate a failure, we stopped the Ethernet interface on the primary machine, using the command below:

```
ifconfig en0 down
```

This causes the standby server to failover. We can see that the backup machine has become the active server by running the **dscontrol high status** command again, as shown in Example 5-21.

Example 5-21 High availability status on the backup server

```
# dscontrol high status
```

High Availability Status:

```
Role ..... Backup
Recovery strategy .... Auto
State ..... Active
Sub-state ..... Not Synchronized
Primary host ..... 10.20.10.102
```

```
Port ..... 12345
Preferred target ..... n/a

Heartbeat Status:
-----
Count ..... 1
Source/destination ... 10.20.10.104/10.20.10.102

Reachability Status:
-----
Count ..... 1
Address ..... 10.20.10.1 reachable
```

Note that this Dispatcher server is now active, but the Sub-state is Not Synchronized (because it cannot communicate with the primary server).

If you look at the IP addresses on the network interfaces of the backup machine, you will notice that this machine now has an IP alias on the Ethernet interface for the cluster IP address, as shown in Example 5-22.

Example 5-22 Network interfaces in the backup server after a failover

```
# ifconfig -a
en0:
flags=4e080863,80<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST,GROUPRT,64BIT,PSEG,CHAIN>
    inet 10.20.10.104 netmask 0xffffffff broadcast 10.20.10.255
    inet 10.20.10.100 netmask 0xffffffff broadcast 10.20.10.255
lo0:
flags=e08084b<UP,BROADCAST,LOOPBACK,RUNNING,SIMPLEX,MULTICAST,GROUPRT,64BIT>
    inet 127.0.0.1 netmask 0xff000000 broadcast 127.255.255.255
    inet6 ::1/0
    tcp_sendspace 65536 tcp_recvspace 65536
```

We kept sending HTTP requests to the cluster address while we were doing the high availability tests to make sure that there is no interruption in the communication between the client and the balanced Web servers.

Tests with manual recovery strategy

If you configured the high availability recovery strategy as manual, you start your tests by forcing the standby server to take over. To do so, go to the standby server, and run the following command:

```
dscontrol high takeover
```

You can also issue a takeover using the Load Balancer GUI. Open the GUI and connect to the standby server. Right-click **High Availability** and select **Takeover**.

Note: You can only run the `dscontrol high takeover` command if the following three conditions are met:

1. The recovery strategy is manual.
2. The machine state is Standby.
3. The Sub-state is Synchronized.

You can confirm this information by first running the `dscontrol high status` command.

5.4 Load Balancer: NAT scenario

In this scenario we use the same servers that we worked with in 5.2, “Load Balancer configuration: basic scenario” on page 135, but now we use NAT/NAPT as the forwarding method instead of MAC forwarding. We assume an unconfigured Load Balancer for this scenario. So if you have tested the basic MAC forwarding scenario first, then you need to delete the existing configuration.

1. Start dsserver, connect to the Dispatcher server and start Executor as explained in steps 1 on page 136 through 5 on page 137.
2. After starting Executor, click **Executor: 10.20.10.102** in the left pane of the GUI window, and locate the Client gateway address field in the right pane (see Figure 5-46 on page 182). This field needs to be filled in order to enable NAT/NAPT and CBR forwarding methods. This is the IP address of the gateway router which handles traffic from the cluster back to the client browser.

In our scenario, where the Web servers are connected to the same network as the Dispatcher server, we used the default gateway IP address in our configuration, which is 10.20.10.1.

Important: We are using 10.20.10.1 because we use the same IP address for inbound and outbound traffic. In case you are not using a common IP address, then you need to specify the IP address of the router which serves as the first hop on the way from the Load Balancer to the client.

Enter the correct IP address and click the **Update Configuration** button.

General settings	
Nonforwarding address:	10.20.10.102
Client gateway address:	10.20.10.1
FIN timeout:	60
High Availability timeout:	2
Wide area network port number:	0
Shared bandwidth (KBytes):	0
Maximum number of clusters:	100
Default maximum ports per cluster:	8

Port-specific settings	
Default maximum servers per port:	32
Default port stale timeout (seconds):	300
Default sticky time (seconds):	0
Default port weight bound:	20
Default port protocol:	TCP/UDP

Update Configuration

Figure 5-46 Configuring the client gateway address

3. After configuring the client gateway address we need to configure the cluster. Right-click **Executor: 10.20.10.102** in the left pane of the GUI and select **Add Cluster...**, as shown in Figure 5-47 on page 183.

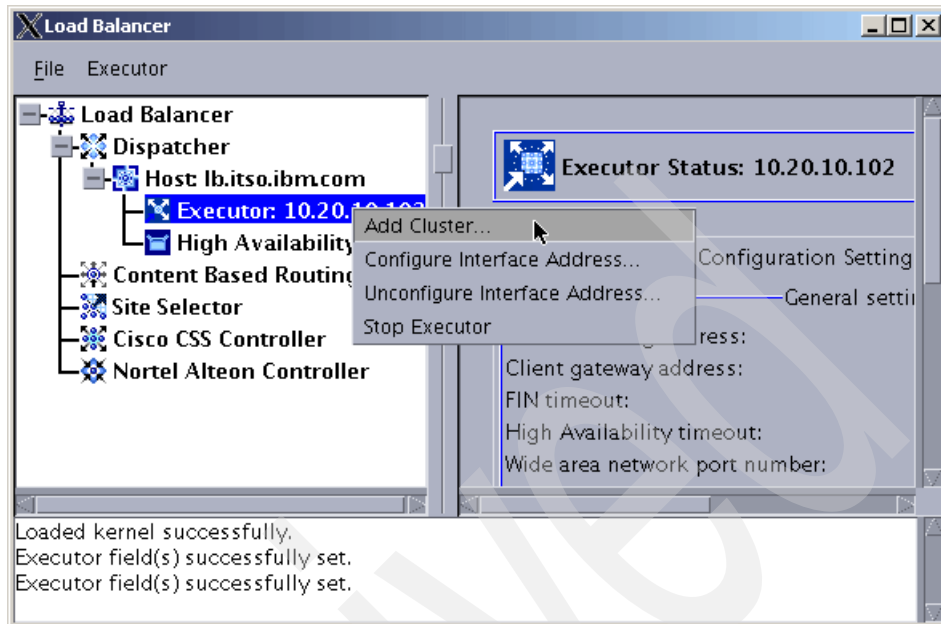


Figure 5-47 Add a cluster - NAT configuration

The information provided for the cluster creation is similar to what we used in the basic scenario (see steps 6 on page 138 through 8 on page 140), details are shown in Figure 5-48.

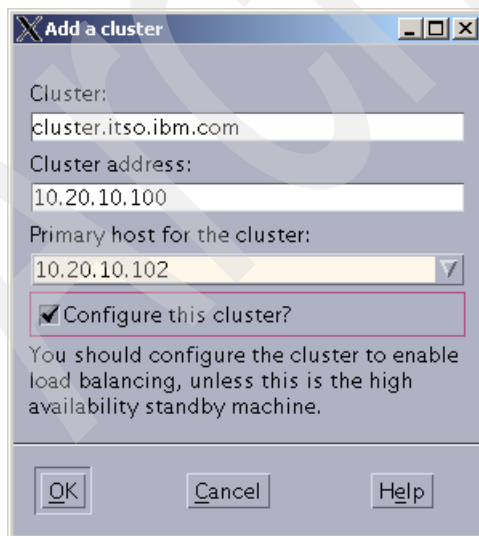


Figure 5-48 Cluster information

If you selected the **Configure this cluster?** checkbox, you need to also provide the information to add the IP alias, as shown in Figure 5-49.

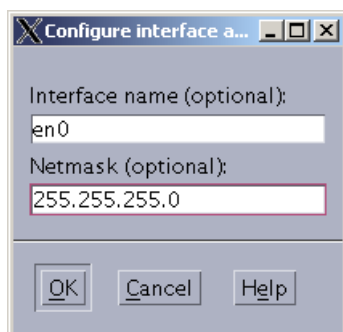


Figure 5-49 Configuring the interface

4. Right-click **Cluster: cluster.itso.ibm.com** in the left pane and select **Add Port...**, as shown in Figure 5-50.

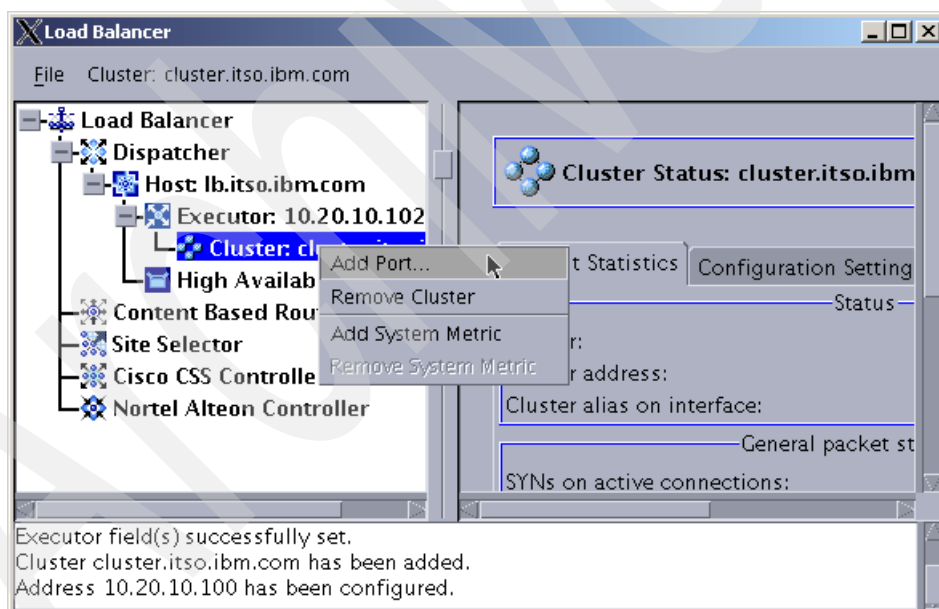


Figure 5-50 Add a port

A pop-up window is displayed similar to the one we had in the basic scenario. But this time, you have all options available in the Forwarding method field.

Type 80 into the Port number field, select **NAT / NAPT** as the Forwarding method and click **OK**. See Figure 5-51 on page 185.

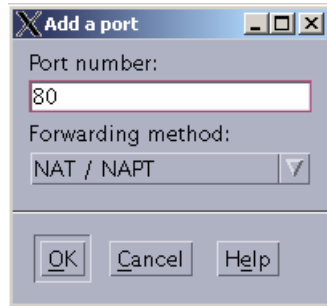


Figure 5-51 Adding a port - NAT forwarding method

5. Right-click **Port: 80** in the left pane and select **Add Server...** as shown in Figure 5-52.

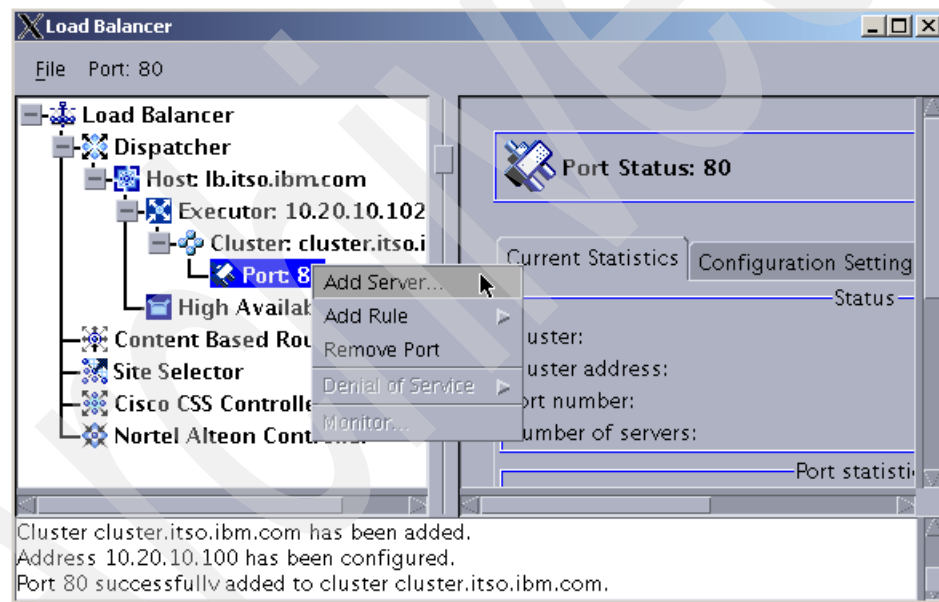


Figure 5-52 Add a server

A pop-up window is displayed, but it contains more fields compared to the one we got when configuring the basic scenario with MAC forwarding.

The two extra fields are the return address and the network router address.

The return address is an extra IP address that is used by Dispatcher as the source IP address of the packets that are sent to the Web servers. You cannot use neither the cluster address nor the NFA (non-forwarding address) as the return address, so you need one additional IP address for this configuration.

The network router address is the router which serves as the first hop on the way from the Load Balancer to the load balanced server. With our single subnet scenario, it is the same address as the client gateway. This field is provided in the server configuration in case you have several balanced servers spread in different remote networks, and you need a different router IP address to reach each server.

Note: When you configure NAT/NAPT on a collocated server, you need to use the local IP address as the network router address. This tells Dispatcher that the desired server is collocated on the local machine.

We added our first Web server as shown in Figure 5-53. We used the same values we had used in the basic scenario (see Figure 5-17 on page 144) and we also provided the IP address we selected as our **Return address** and our default gateway IP address as our **Network router address**.

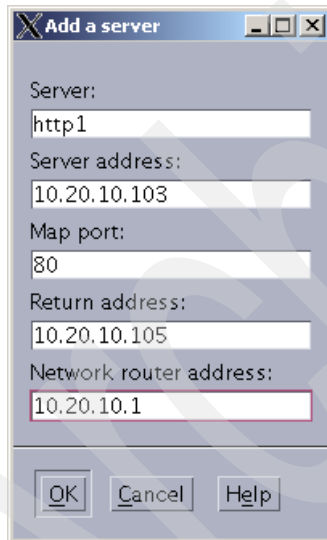


Figure 5-53 Adding the first balanced server

6. Add the second Web server. We used the same values for our second Web server as the ones we used in the basic scenario, and we also used the same return address and the same network router address, as shown in Figure 5-54 on page 187.

Figure 5-54 Adding the second balanced server

7. Follow steps 11 on page 144 through step 12 on page 146 to start Manager, advisors and save the configuration.
8. The last thing you need to configure is to add the return address to the operating system, so it is able to handle the responses from the balanced servers. You can either use the GUI or the command line to do this.

Important: It is sufficient to configure the return address if you are not in a collocated environment. If, however, one or both of your Load Balancer servers are collocated, you need to perform additional steps. These differ for each operating system, so refer to the Load Balancer Administration Guide, Chapter 21, Section “Using collocated servers - Configuring server collocation with Dispatcher’s nat forwarding” for complete instructions for your operating system.

- a. In the GUI, right-click **Executor** and select **Configure Interface Address...** Enter the return address into the Interface address field (in our example this is 10.20.10.105), type the interface name into the optional Interface name field (in our example this is en0) and enter the netmask into the Netmask field (in our example this is 255.255.255.0). See Figure 5-55 on page 188.

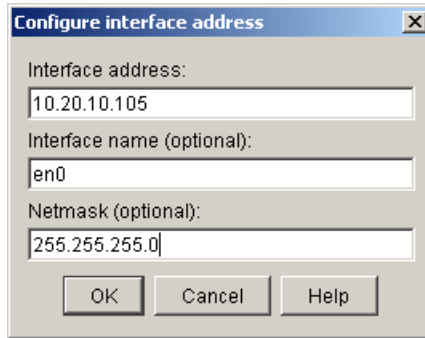


Figure 5-55 Configure the return address

- a. Alternatively, if you prefer the command line interface, run one of the following commands:

```
dscontrol executor configure 10.20.10.105 en0 255.255.255.0 (AIX)
dscontrol executor configure 10.20.10.105 en0 255.255.255.0 (Windows)
dscontrol executor configure 10.20.10.105 eth0:1 255.255.255.0 (Linux)
```

9. Do not forget to save your configuration file.

Example 5-23 shows the default.cfg configuration file for this scenario (remember each command must be on one line):

Example 5-23 Configuration file (default.cfg) for NAT scenario

```
dscontrol set loglevel 1
dscontrol executor start
dscontrol executor set clientgateway 10.20.10.1

dscontrol cluster add cluster.itso.ibm.com address 10.20.10.100 primaryhost
10.20.10.102
dscontrol cluster set cluster.itso.ibm.com proportions 49 50 1 0
dscontrol executor configure 10.20.10.100 en0 255.255.255.0

dscontrol port add cluster.itso.ibm.com:80 method nat reset no
dscontrol port set cluster.itso.ibm.com:80 porttype tcp

dscontrol server add cluster.itso.ibm.com:80:http1 address 10.20.10.103 router
10.20.10.1 returnaddress 10.20.10.105
dscontrol executor configure 10.20.10.105 en0 255.255.255.0

dscontrol server add cluster.itso.ibm.com:80:http2 address 10.20.10.104 router
10.20.10.1 returnaddress 10.20.10.105
dscontrol executor configure 10.20.10.105 en0 255.255.255.0
```

```
dscontrol manager start manager.log 10004
```

```
dscontrol advisor start Http 80 Http_80.log
```

Note: You do not need to configure the balanced Web servers when using the NAT/NAPT forwarding method as described in 5.2.2, “Configuring the balanced servers” on page 148. This step is only necessary when using the MAC forwarding method.

5.4.1 Testing the NAT scenario

The tests we performed were similar to the ones we did in 5.2.3, “Testing the basic scenario” on page 156.

We generated requests to the cluster using Rational Load Tester and monitored the load distribution using the port Monitor, as shown in Figure 5-56.

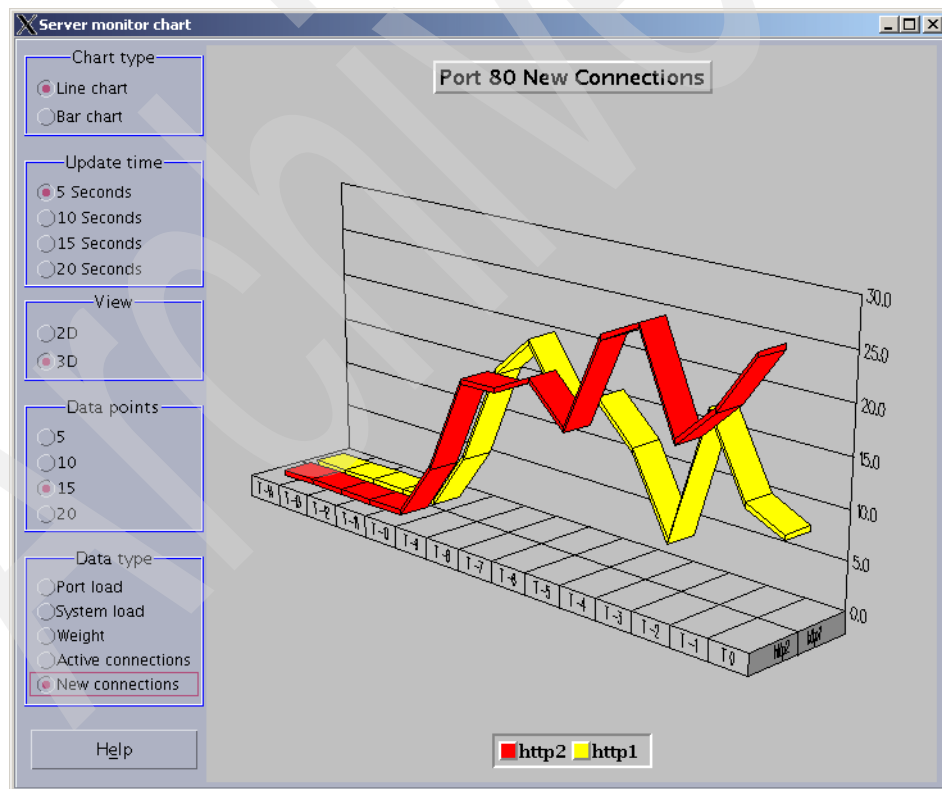


Figure 5-56 Monitoring the NAT scenario test

5.5 Load Balancer: additional configuration options

This section describes some additional configuration and usage options that you might want to add to your setup. We cover the following:

- ▶ “Basic Load Balancer scenario with customizable advisor settings” on page 190.
- ▶ “Using WebSphere Application Server custom advisor” on page 193.
- ▶ “Starting Dispatcher automatically after a reboot” on page 203.
- ▶ “Starting and stopping Dispatcher components” on page 204.

5.5.1 Basic Load Balancer scenario with customizable advisor settings

The advisor used in our basic scenario tested the connectivity to the Web server by sending the following request:

```
HEAD / HTTP/1.0
```

This request is served by the Web server default index page. There are certain situations where this request will be inappropriate:

- ▶ If the Web server cannot serve a default page for URI “/”; this causes the request to fail, although the actual application is running fine in the back-end application servers.
- ▶ If the Web server plug-in fails, the Web server can still respond to this request if it is a local page, but it will not be able to communicate with the application servers. Any request sent to this Web server which requires communication with the back-end application servers will fail.

We have two options to solve this problem: we can customize the request that is sent by the advisor, so we make sure the request tests the Web server plug-in and the back-end application server, or we can write a custom advisor.

We first explain how to customize the advisor request, and later in 5.5, “Load Balancer: additional configuration options” on page 190 we explain the custom advisors.

Using the basic scenario that was set up in 5.2, “Load Balancer configuration: basic scenario” on page 135, we now change the advisor request so it uses an application request: /w1m/BeenThere.

1. Open the Load Balancer GUI and connect to the Dispatcher server.
2. Select **http1** from the left pane (on Port:80).

3. In the right pane, click the **Configuration Settings** tab (see Figure 5-57).
4. Enter the following into the HTTP Advisor Request field:
HEAD /wlm/BeenThere HTTP/1.0
Click the **Update Configuration** button.
5. In the HTTP Advisor Response field type:
200
Click the **Update Configuration** button.

General settings	
Collocated:	no
Current status:	up
Server weight:	set by manager
Fixed weight:	no
Sticky for rule:	yes
Network router address:	n/a
HTTP Advisor Request:	HEAD /wlm/BeenThere HTTP/1.0
HTTP Advisor Response:	200

Update Configuration

Figure 5-57 Customizing the advisor request

6. Repeat steps 2 through 5 for each Web server.

You can check if the advisor is receiving responses for the new request in the **dscontrol manager report** command, Monitor (select the **Port load** data type), or you can click **Advisor: Http 80** in the left pane of the GUI and check the Server Statistics in the right pane as shown in Figure 5-58 on page 192.

Current Statistics

Configuration Settings

Status

Advisor name:

Http

Port number:

80

Log file name:

Http_80.log

Server statistics

Cluster	Server	Load
cluster.itso.ibm.com	http2	37
cluster.itso.ibm.com	http1	44

Refresh Statistics

Figure 5-58 Advisor statistics

Check the access log of each Web server, you should see the entries shown in Example 5-24 which correspond to the advisor requests.

Example 5-24 Web server access log

```

10.20.10.102 - - [22/Oct/2004:16:05:50 -0400] "HEAD /wlm/BeenThere HTTP/1.0" 200 -
10.20.10.102 - - [22/Oct/2004:16:06:02 -0400] "HEAD /wlm/BeenThere HTTP/1.0" 200 -
10.20.10.102 - - [22/Oct/2004:16:06:10 -0400] "HEAD /wlm/BeenThere HTTP/1.0" 200 -
10.20.10.102 - - [22/Oct/2004:16:06:23 -0400] "HEAD /wlm/BeenThere HTTP/1.0" 200 -
10.20.10.102 - - [22/Oct/2004:16:06:30 -0400] "HEAD /wlm/BeenThere HTTP/1.0" 200 -
10.20.10.102 - - [22/Oct/2004:16:06:37 -0400] "HEAD /wlm/BeenThere HTTP/1.0" 200 -
10.20.10.102 - - [22/Oct/2004:16:06:45 -0400] "HEAD /wlm/BeenThere HTTP/1.0" 200 -
10.20.10.102 - - [22/Oct/2004:16:06:57 -0400] "HEAD /wlm/BeenThere HTTP/1.0" 200 -
10.20.10.102 - - [22/Oct/2004:16:07:28 -0400] "HEAD /wlm/BeenThere HTTP/1.0" 200 -

```

Example 5-25 shows the commands that you can use to set the same parameters using the command line interface. Please note that each individual command has to be on one line in the configuration file.

Example 5-25 Changes in the configuration file

```

dscontrol server set cluster.itso.ibm.com:80:http2 advisorrequest "HEAD
/wlm/BeenThere HTTP/1.0"
dscontrol server set cluster.itso.ibm.com:80:http2 advisorresponse "200"

dscontrol server set cluster.itso.ibm.com:80:http1 advisorrequest "HEAD
/wlm/BeenThere HTTP/1.0"
dscontrol server set cluster.itso.ibm.com:80:http1 advisorresponse "200"

```

5.5.2 Using WebSphere Application Server custom advisor

The WebSphere custom advisor must be considered as a monitoring extension associated with each HTTP server on the cluster. The custom advisor prevents requests from being sent to a specific HTTP server when this HTTP server cannot appropriately fulfill them (for example, when the WebSphere server it sends requests to is down).

A sample custom advisor for WebSphere Application Server is included in the `<install_path>/servers/samples/CustomAdvisors` directory.

Two files are used to implement the advisor:

- ▶ **ADV_was.java:** The custom advisor code that should be compiled and executed on the Dispatcher machine.
- ▶ **LBAdvisor.java.servlet:** The servlet code (it must be renamed to `LBAdvisor.java`) that should be compiled and installed on the WebSphere Application Server machine.

The servlet returns the following string in case of success:

```
LBAdvisor/0.92 100  
Thu Nov 04 10:06:59 EST 2004
```

The advisor parses the string to get the WebSphere status (100) and informs the Dispatcher monitor.

Restriction: Using the WebSphere Application server custom advisor is only useful in environments where you are *not* using the WebSphere workload management options - provided by the plug-in in the Web server. When we use WebSphere workload management, the requests from multiple HTTP servers are sent to a group of application servers, distributed also among multiple machines. We cannot associate the service layer provided by the application server to an HTTP server anymore, since the plug-in is responsible for distributing the requests.

So the custom advisor could be used, for example, if you have enabled server affinity on the Load Balancer, then forward requests from one Web server to a certain application server and the related database. In such a configuration, the advisor can accurately mark down or lower the weight based on the full test - including the application server and the database.

When using WebSphere plug-in workload management, as in our sample topology, monitoring the HTTP servers using the HTTP advisor is probably the best choice.

Edit the sample files and prepare the advisor and the servlet to be installed on the proper machines as follows:

1. Rename the file ADV_was.java to ADV_was6.java, because was6 will be the name of the advisor.
2. Modify the Java file ADV_was6.java to reflect your settings.

- a. To reflect the new class name, the line:

```
public class ADV_was extends ADV_Base implements ADV_MethodInterface
{
```

must be changed to:

```
public class ADV_was6 extends ADV_Base implements ADV_MethodInterface
{
```

and the line:

```
public ADV_was() {
```

must be changed to:

```
public ADV_was6() {
```

- b. To reflect the new advisor name, the line:

```
static final String ADV_WAS_NAME = "was";
```

must be changed to:

```
static final String ADV_WAS_NAME = "was6";
```

- c. To reflect the response of IBM HTTP Server on AIX, the line:

```
static final String HTTP_GOOD = "HTTP/1.1 200 ok";
```

must be changed to:

```
static final String HTTP_GOOD = "HTTP/1.1 200 OK";
```

- d. To reflect the location, where you have deployed the LBAvisor servlet, the line:

```
static final String SERVLET_LOCATION = "/servlet/LBAvisor";
```

must be changed to:

```
static final String SERVLET_LOCATION = "/advisor/servlet/LBAvisor";
```

3. Compile the advisor using the following command (the command must be typed on one single line):

```
javac -classpath
/opt/ibm/edge/lb/servers/lib/ibmlb.jar:/opt/ibm/edge/lb/admin/lib/j2ee.jar
ADV_was6.java
```

Make sure that the compilation finishes without errors.

4. Copy the ADV_was6.class file to the <LB_install_path>/servers/lib/CustomAdvisors directory:

```
cp ADV_was6.class /opt/ibm/edge/lb/servers/lib/CustomAdvisors
```

The advisor is now installed, but you still need to compile and deploy the advisor servlet, which will run in the back-end application servers.

We used Application Server Toolkit (AST) to edit and compile the advisor servlet, and to create the WAR file. The structure of the application we created for the LBAdvisor is shown in Figure 5-59.

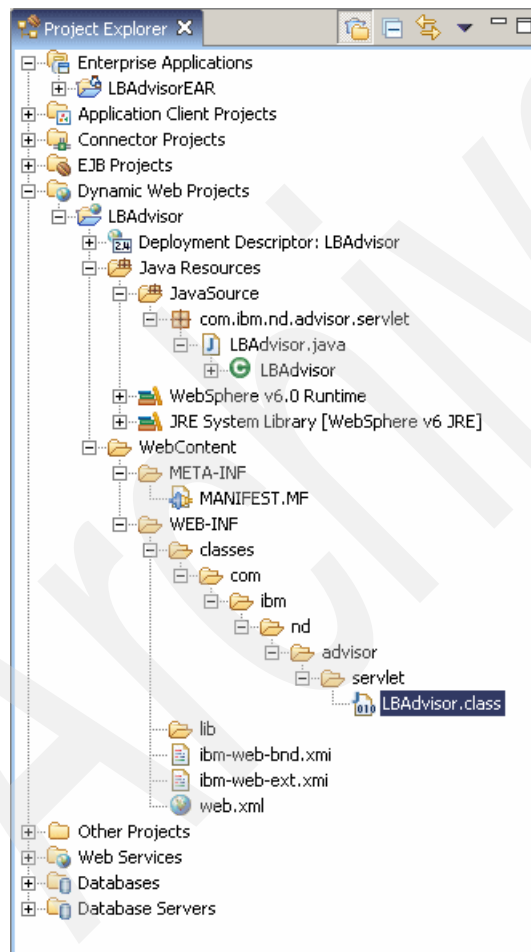


Figure 5-59 LBAdvisor in AST

The provided sample performs a select in the sample database. In order to use this sample with our scenario (we are using the Trade 6 database for this test), we made the following changes to the servlet LBAAdvisor.java using AST:

Important: The following changes assume that you installed the tradedb with the user “Administrator”, and that the database uses the schema “Administrator” for the tables. If you installed the database and created the tables with the default user (db2admin in Windows and db2inst1 in Linux/UNIX) you must change the code accordingly.

- Change the syntax of the sleep method call from:

```
try {  
    _checker.sleep(_interval * 1000);  
} catch (Exception ignore) { }
```

to:

```
try {  
    Thread.sleep(_interval * 1000);  
} catch (Exception ignore) { }
```

- Change the reference to the database from:

```
private static String _dataSourceName = "jdbc/sample";
```

to:

```
private static String _dataSourceName = "java:comp/env/jdbc/advisordb";
```

- Change the database user ID from:

```
private static String _dbUserid = "khygh";
```

to the proper one in your environment. In our scenario we used:

```
private static String _dbUserid = "Administrator";
```

- Change the password of the database user ID from:

```
private static String _dbPassword = "one4all";
```

to the proper one in your environment. In our scenario we used:

```
private static String _dbPassword = "my_password";
```

- Change the database schema name from:

```
private static String _dbOwner = "khygh";
```

to:

```
private static String _dbOwner = "ADMINISTRATOR";
```

- Change the select statement from:

```
private static final String _query1 = "Select FirstNme from ";
```

```
private static final String _query2 = ".Employee where LASTNAME =
'PARKER'";
```

to:

```
private static final String _query1 = "Select EMAIL from ";
private static final String _query2 = ".accountprofileejb where USERID =
'uid:0'";
```

Note that we rewrote the select in the second line to use the USERID uid:0. This user is created by default in the Trade 6 database. If you deleted this user, change this line and insert a user that exists in your database. Be careful to use the correct capitalization for the user ID. You can also customize the whole select statement.

- Change the following lines from:

```
Hashtable parms = new Hashtable(2);
parms.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");
parms.put(Context.PROVIDER_URL, "iiop:///"); // local machine
Context context = new InitialContext(parms);
```

to:

```
Context context = new InitialContext();
```

Note that we deleted the first three lines from the original code.

- Add a servlet mapping to the Web module deployment descriptor (the web.xml file):

Example 5-26 Servlet mapping for LBAdvisor

```
<servlet>
  <display-name>
    LBAdvisor</display-name>
  <servlet-name>LBAdvisor</servlet-name>
  <servlet-class>
    com.ibm.nd.advisor.servlet.LBAdvisor</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>LBAdvisor</servlet-name>
  <url-pattern>/servlet/LBAdvisor</url-pattern>
</servlet-mapping>
```

- Add a resource reference to the Web module deployment descriptor (the web.xml file):

Example 5-27 Resource reference for LBAdvisor application

```
<resource-ref id="ResourceRef_1099586856129">
  <res-ref-name>jdbc/advisordb</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
```

```
<res-auth>Container</res-auth>
<res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
```

We then exported the application to a WAR file, and installed it on our sample topology using these steps:

1. Create two application servers, Advisor1 on app1Node and Advisor2 on app2Node. For more information about creating application servers refer to *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.
2. Create a new datasource with the same parameters (except for the JNDI name which should be jdbc/advisordb) as the Trade 6 TradeDataSource for your LBAvisor application. You can use the same datasource, but we recommend creating a new one so LBAvisor does not use resources from the Trade 6 datasource.

Refer to the InfoCenter for more information about how to create a datasource. The InfoCenter is available at:

<http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp>

3. Select **Applications -> Enterprise Applications -> Install New Application**. Select **LBAvisor.war** (the file that was generated previously using AST) and enter `/advisor` in the Context root field, as shown in Figure 5-60. Click **Next**.

The screenshot shows a dialog box titled "Preparing for the application installation". It contains a section "Path to the new application." with two radio buttons: "Local file system" (selected) and "Remote file system". Under "Local file system", there is a "Specify path" label, a text box containing "C:\temp\LBAvisor.war", and a "Browse..." button. Under "Remote file system", there is a "Specify path" label, an empty text box, and a "Browse..." button. Below these sections is a "Context root" label, a text box containing "/advisor", and a note "Used only for standalone Web modules (.war files)". The "Context root" text box is circled in red. At the bottom are "Next" and "Cancel" buttons.

Figure 5-60 Selecting LBAvisor.war for installation

4. In the next pane select **Use the virtual host name for Web modules**, enter the virtual host **default_host** and click **Next**.

- Click **Continue**. In the Step 1 pane, enter **LBAdvisor_war1** in the application name field and click **Next**.
- In the Step 2 window, select the Web server **http1** and the application server **Advisor1** under Clusters and Servers, select the **LBAdvisor** module and click **Apply**, as shown in Figure 5-61. Then click **Next**.

Figure 5-61 Mapping the LBAdvisor module to servers

In this case, http1 will only forward requests to the application server Advisor1. If anything fails in the application itself, or in the communication of this application server to the database, the advisor will detect it and set its port value of http1 to -1.

- In the next pane you need to associate the resource reference to the datasource. You can select the datasource used by Trade 6 (jdbc/TradeDataSource) or the one you created in step 2 on page 198 (we named our datasource /jdbc/advisordb).

Make sure you also map the correct authentication method (TradeDataSourceAuthData), as shown in Figure 5-62. Click **Next**.

Select	Module	EJB	URI	Reference binding	JNDI name	Login configuration
<input type="checkbox"/>	LBAdvisor		LBAdvisor.war,WEB-INF/web.xml	jdbc/advisordb	jdbc/advisordb	Resource authorization: Container Authentication method: DefaultPrincipalMapping TradeDataSourceAuthData

Figure 5-62 Mapping resources to the LBAdvisor module

- Click **Continue** on the Application Resource Warnings pane, then in the Step 4 window select the virtual host **default_host**. Click **Next**.

9. Click **Finish** on the Summary pane.
10. Repeat steps 3 on page 198 through 9. This time, however, you use **LBAdvisor_war2** as the application name in step 5 on page 199, and you map it to the **http2** Web server and **Advisor2** application server.
11. Save the configuration, update and propagate the Web server plug-in, then start the application servers (in case they are stopped) and the applications LBAdvisor_war1 and LBAdvisor_war2. See Figure 5-63.

<div> <div>Start</div> <div>Stop</div> <div>Install</div> <div>Uninstall</div> <div>Update</div> <div>Rollout Update</div> <div>Remove File</div> <div>Export</div> <div>Export DDL</div> </div>		
<div> <div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> </div>		
Select	Name	Status
<input type="checkbox"/>	BeenThere	
<input type="checkbox"/>	DynaCacheEsi	
<input type="checkbox"/>	Dynamic Cache Monitor	
<input type="checkbox"/>	LBAdvisor_war1	
<input type="checkbox"/>	LBAdvisor_war2	
<input type="checkbox"/>	Trade	
Total 6		

Figure 5-63 LBAdvisor_war1 and LBAdvisor_war2 applications

Note that the environment we just created does not use WebSphere workload management: if either the application server Advisor1 or the Web server http1 fails, http1 will be marked down, so all requests will be forwarded to http2. The same thing will happen if either Advisor2 or http2 fails: http2 will be marked down.

12. Test if the servlet is responding using `http://http1/advisor/servlet/LBAdvisor?debug=1`. This URL returns a HTML/text page with some diagnostic information, as shown in Figure 5-64 on page 201.

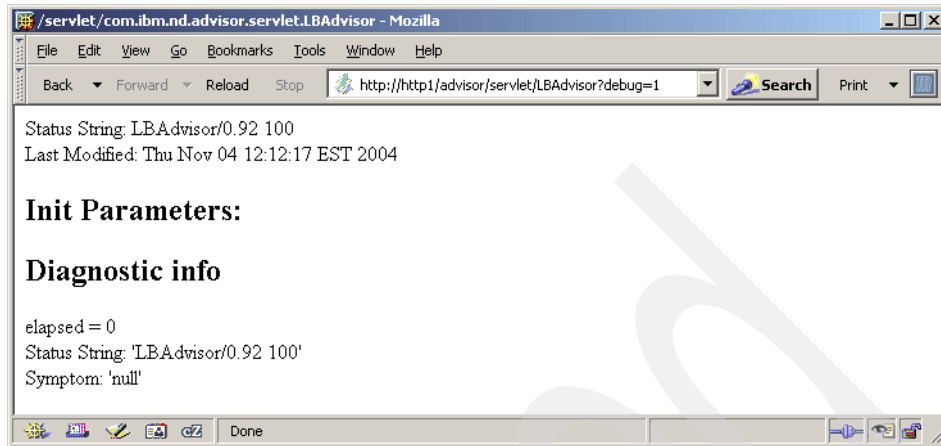


Figure 5-64 LBAdvisor diagnostic information

13..Restart the Load Balancer server:

```
dsserver stop
dsserver start
```

14.Start the custom advisor using the command:

```
dscontrol advisor start was6 80
```

where was6 is your custom advisor name and 80 is the port number where the advisor opens a connection with the target server.

Restriction: You can only start one advisor for a certain port. Therefore, if you have the default HTTP Advisor started, you first need to stop it.

15.Start the GUI of Load Balancer and select your new advisor, then select the **Current Statistics** tab on the right. The advisor status is displayed. Click **Refresh** to update the status. If the advisor is working properly, the load value is displayed. If there is no response, a status of -1 is returned (see Figure 5-65 on page 202).

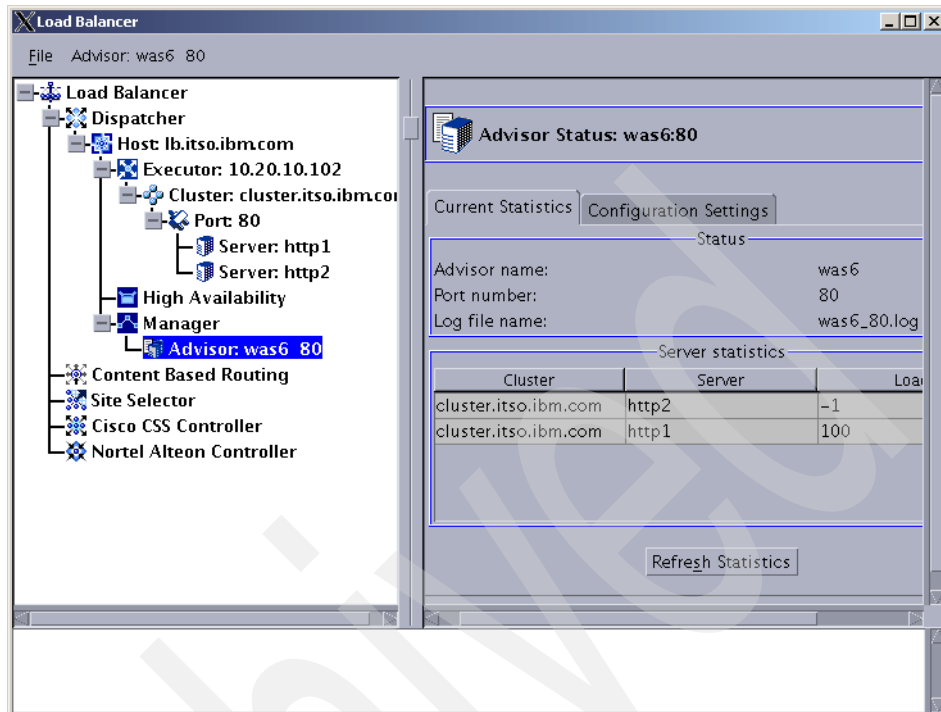


Figure 5-65 Advisor statistics

If there is no response, one option is to select the **Configuration Settings** pane and change the advisor logging level to Advanced, for example, then look at the log file at <LB_install_path>/servers/logs/dispatcher. A lot of detailed information is recorded, showing all requests and responses, which may help you locate where the problem is occurring.

If we simulate a failure by stopping the application server Advisor2 or causing a failure in the communication to the database server, we can see how the requests are directed to the available application server (associated with the http1 server). We can see the graphical statistics of the traffic by selecting the **Monitor** option under the cluster port the advisor is associated with, as shown in Figure 5-66 on page 203.

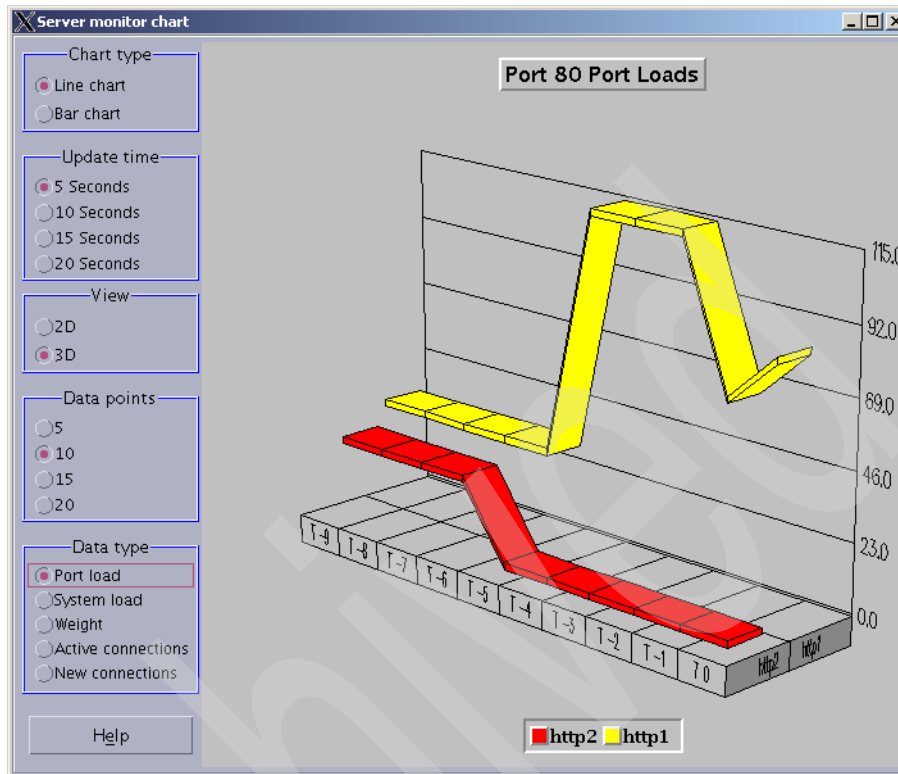


Figure 5-66 Server monitor chart showing http2 is down

5.5.3 Starting Dispatcher automatically after a reboot

If you are running Dispatcher on a Windows system, the IBM Dispatcher service that was created during the product installation is configured to be automatically started after each reboot.

If you are running Dispatcher on a UNIX system, you must configure the system to run the **dsserver** command after each reboot. Make sure your configuration filename is default.cfg, because when dsserver is run it automatically loads the default.cfg configuration file.

In our AIX environment, we enabled the automatic startup of dsserver. We added it to the inittab by running the following command:

```
mkitab "ds:2:wait:/usr/bin/dsserver > /dev/console 2>&1"
```

Tip: You can also use other scripts that are run during the server start up process, for example `/etc/rc.tcpip` (in AIX systems) or `/etc/rc.local` (in Linux systems). Make sure you consult the system administrator to find the most suitable option for your environment.

5.5.4 Starting and stopping Dispatcher components

Each component has a start and stop command available. To stop all components in the Dispatcher server use the following commands:

```
dscontrol executor stop
dsserver stop
```

Note: If you run only **dsserver stop**, Executor continues running, so the load balancing mechanism is still active. When you intend to fully stop Dispatcher you need to first stop Executor and then stop dsserver.

In Windows systems, stopping the IBM Dispatcher service only stops the administration server. You need to stop Executor manually before stopping the service.

To start Dispatcher, simply run **dsserver** (or start the Windows service called IBM Dispatcher) and it automatically loads the configuration file `default.cfg`, which contains the start commands for all other necessary components.

5.6 Caching Proxy installation

As mentioned in 5.1, “Load Balancer installation” on page 128, you can install the WebSphere Edge Components products using the common wizard or the operating system tools.

We first describe the installation in a Windows 2000 server using the wizard, and later we describe the installation in an AIX server using SMIT (a management tool provided by the operating system).

Before starting the installation, refer to *Concepts, Planning, and Installation for Edge Components Version 6.0*, GC31-6855 for the prerequisites and supported operating systems.

Important: In our tests, the installation wizard for AIX systems did not detect the absence of a prerequisite fileset: bos.iocp.rte. It is also not documented in the product's manuals.

Make sure you follow the instructions in 5.6.1, “Checking prerequisites” on page 205 before installing Caching Proxy on an AIX system.

5.6.1 Checking prerequisites

There are several prerequisites you need to verify.

Windows prerequisites

Make sure you have a Java Runtime Environment 1.4.2 (or later) installed on your system and it is in the system path. Caching Proxy does not come with a JRE but the configuration wizard needs it.

AIX prerequisites

AIX installation requires the IOCP device, which is contained in the fileset bos.iocp.rte. Check if this driver is already installed by running the command:

```
lslpp -l bos.iocp.rte
```

If this fileset is installed, you receive an output similar to the one shown in Example 5-28.

Example 5-28 Checking the fileset bos.iocp.rte

# lslpp -l bos.iocp.rte			
Fileset	Level	State	Description

Path: /usr/lib/objrepos			
bos.iocp.rte	5.2.0.10	COMMITTED	I/O Completion Ports API
Path: /etc/objrepos			
bos.iocp.rte	5.2.0.10	COMMITTED	I/O Completion Ports API

If this fileset is not installed, you receive the following message:

```
lslpp: 0504-132 Fileset bos.iocp.rte not installed.
```

This fileset is provided on the AIX installation media, so request the system administrator to install this fileset and reboot the server before proceeding. The reboot is necessary in order to create the iocp device.

Make sure that the iocp device has already been created before starting the Caching Proxy installation. See Example 5-29.

Example 5-29 Checking the iocp device

```
# lsdev -Cc iocp
iocp0 Defined I/O Completion Ports
```

In Example 5-29, the iocp0 device is in Defined state. This state will be changed to Available after the Caching Proxy installation.

Caching Proxy also requires the AIO device, which is contained in the fileset bos.rte.aio. This fileset is automatically installed with the operating system installation. Make sure the aio device is in available state by running the command shown in Example 5-30.

Example 5-30 Checking the aio device

```
# lsdev -Cc aio
aio0 Available Asynchronous I/O (Legacy)
```

If the device is in defined state, run the following command then reboot the server.

```
# chdev -l aio0 -P -a autoconfig='available'
```

5.6.2 Caching Proxy installation wizard

The WebSphere Edge Components installation media provides a wizard for all platforms so the installation is similar for all supported operating systems.

If you are installing Caching Proxy on AIX, make sure you installed the IOCP device as explained in 5.6.1, “Checking prerequisites” on page 205 before you start the installation.

1. Mount the installation media and start LaunchPad by running launchpad.sh (on UNIX servers) or launchpad.bat (on Windows servers).

The LaunchPad window opens as shown in Figure 5-1 on page 129.

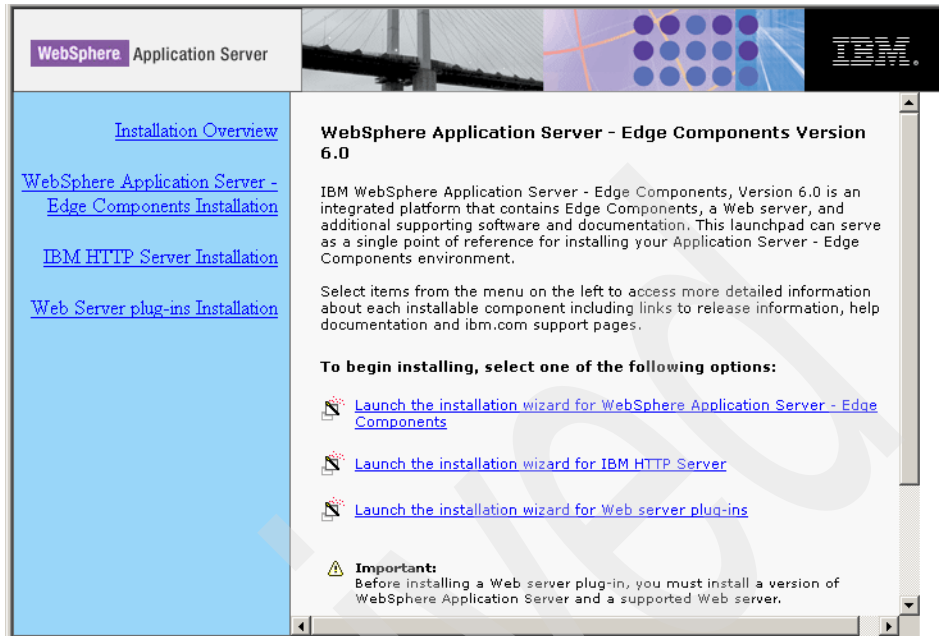


Figure 5-67 LaunchPad window

Click **Launch the installation wizard for WebSphere Application Server - Edge Components**.

2. Click **Next** on the Welcome screen and click **Yes** to accept the product license.
3. In the Component Selection window you select the component you want to install. Select the **Caching Proxy** checkbox, and click the button **Change Subcomponents...** as shown in Figure 5-2 on page 130.

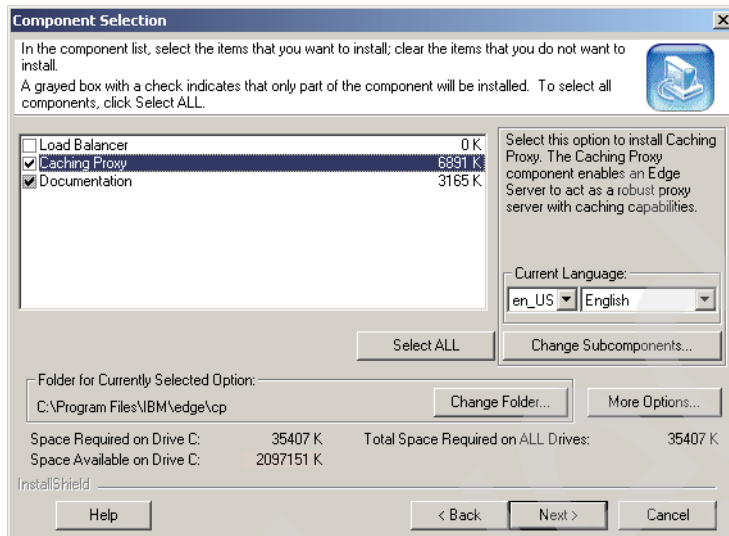


Figure 5-68 Components Selection window

- The Subcomponent Selection window is opened. Select the subcomponents you want to install. The Caching Proxy Base Server subcomponent is mandatory. By default, all subcomponents are selected, as shown in Figure 5-3 on page 131. Click **OK** to return to the Component Selection window.

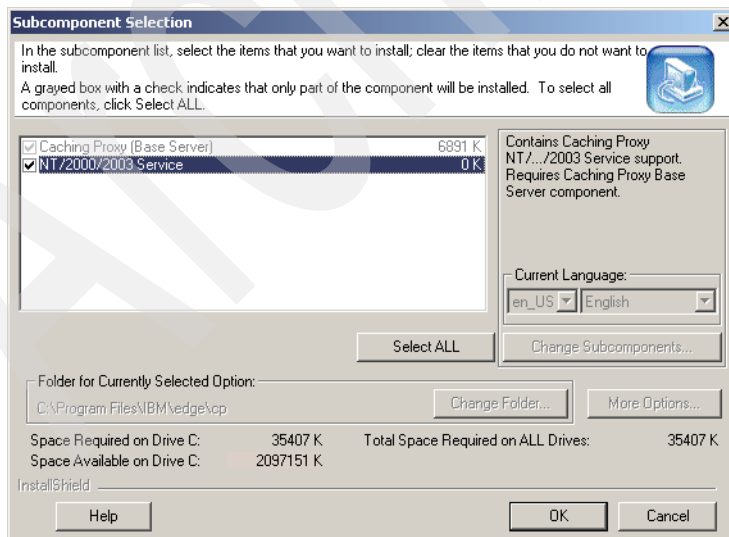


Figure 5-69 Subcomponent Selection window

The default installation path is C:\Program Files\IBM\edge\cp. If you want to install the product into a different path, click **Change Folder** and enter the path. Click **Next** to continue the installation.

5. Make sure that the selected options are correct in the Installation Selection Summary, and click Finish to start the installation, as shown in Figure 5-4 on page 132.

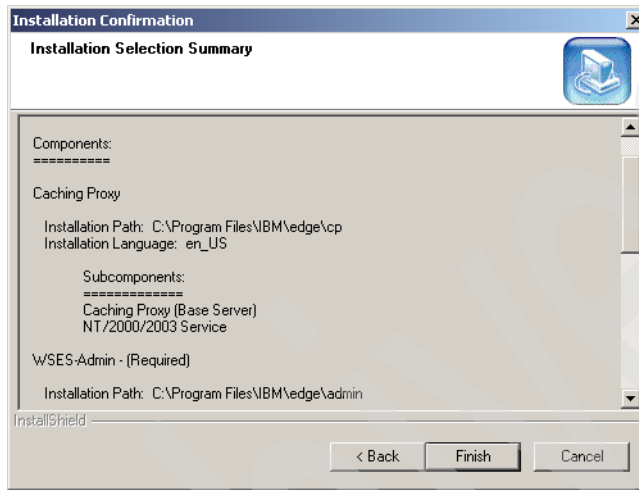


Figure 5-70 Installation Summary window

6. At the end of the installation you have the option to reboot the server. Make sure you do so before using the product.

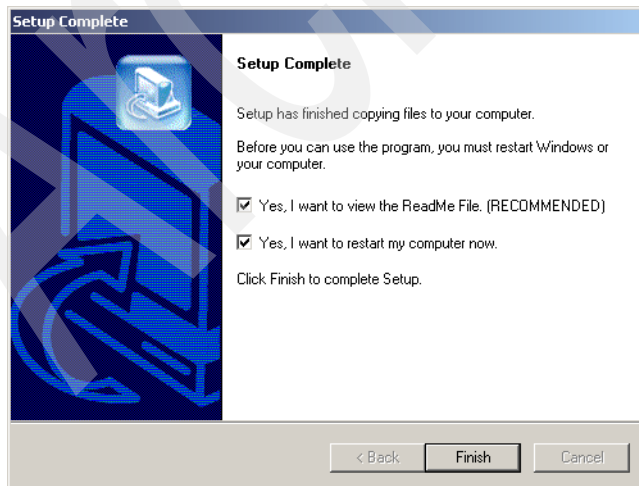


Figure 5-71 Setup Complete window

Important: It is highly recommended that you also install the latest eFixes/PTFs.

5.6.3 Caching Proxy installation using SMIT in AIX

AIX provides a tool to manage the operating system, which is SMIT. In this section we describe how to install Caching Proxy using SMIT.

Make sure you installed the IOCP device before you start the installation. Refer to 5.6.1, “Checking prerequisites” on page 205.

Follow the steps described here in order to install the Caching Proxy.

1. Log in as root.
2. Mount the WebSphere Edge Components installation media and go to the directory you used as mount point.
3. Go to the icu directory:

```
cd icu
```

4. Run the following command:

```
installp -acXd ./wses_icu.rte all
```

A summary is presented at the end of the installation process. Make sure that the fileset was successfully installed, as shown in Example 5-31.

Example 5-31 Fileset wses_icu.rte successfully installed

Installation Summary

Name	Level	Part	Event	Result
wses_icu.rte	6.0.0.0	USR	APPLY	SUCCESS
wses_icu.rte	6.0.0.0	ROOT	APPLY	SUCCESS

5. Go to the admin directory:

```
cd ../admin
```

6. Run the following command:

```
installp -acXd ./wses_admin.rte all
```

A summary is presented at the end of the installation process. Make sure that the fileset was successfully installed, as shown in Example 5-32.

Example 5-32 Fileset wses_admin.rte successfully installed

Installation Summary

Name	Level	Part	Event	Result
wses_admin.rte	6.0.0.0	USR	APPLY	SUCCESS
wses_admin.rte	6.0.0.0	ROOT	APPLY	SUCCESS

7. Go to the cp directory:

```
cd ../cp
```

8. Run the following command:

```
smit install_all
```

9. In the Install and Update from ALL Available Software screen, type the full path to the cp directory mentioned in step 7 or type a period (.) which represents the current directory, as shown in Figure 5-5 on page 133.

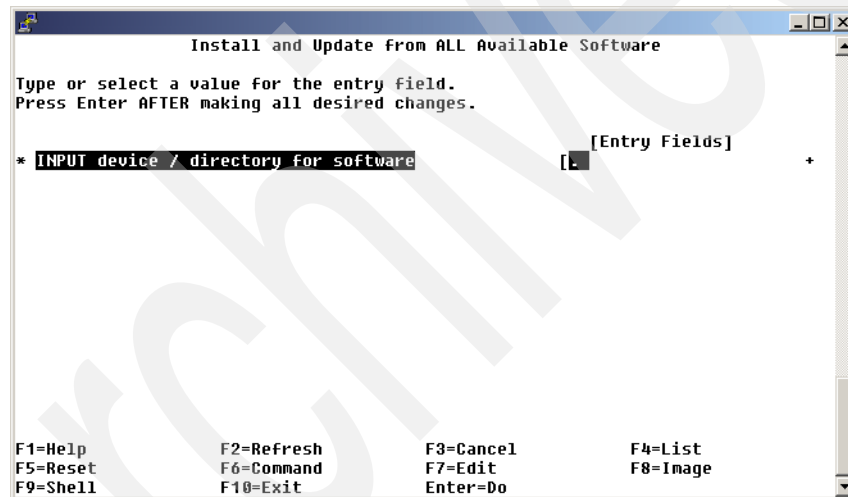


Figure 5-72 Selecting the path of the installation media

10. In the Install and Update from ALL Available Software screen, put the cursor in the SOFTWARE to install field and press the **F4** key (**ESC+4** in VT100 and ASCII emulators).

11. From the SOFTWARE to install list, select the components you want to install. We selected the following components:

- Caching Proxy (wses_cp.base)
- Caching Proxy Messages - en_US (wses_cp.msg.en_US.base)

You may need to scroll the screen to the left using the right arrow key in order to see the fileset names listed in parenthesis.

After you finish selecting the filesets to install, press the **Enter** key.

12. Back at the Install and Update from ALL Available Software screen, select **yes** in the ACCEPT new license agreements? field as shown in Figure 5-73, and press **Enter** to start the installation.

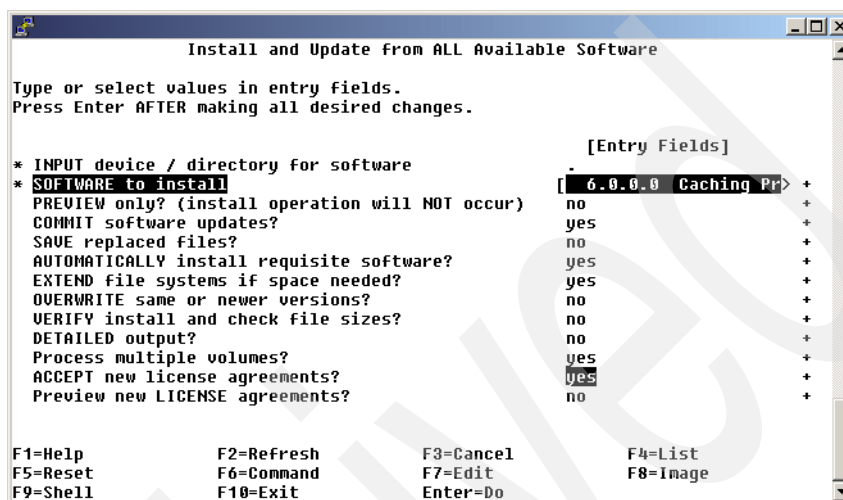


Figure 5-73 SMIT installation screen

13. When the installation finishes, check the installation summary to make sure that all filesets were installed successfully, as shown in Example 5-33.

Example 5-33 Installation summary for Caching Proxy

Installation Summary

Name	Level	Part	Event	Result
wses_cp.msg.en_US.base	6.0.0.0	USR	APPLY	SUCCESS
wses_cp.msg.en_US.base	6.0.0.0	ROOT	APPLY	SUCCESS
wses_cp.base	6.0.0.0	USR	APPLY	SUCCESS
wses_cp.base	6.0.0.0	ROOT	APPLY	SUCCESS

5.7 Caching Proxy configuration

After installation, Caching Proxy is ready to use and it operates by default as a forward proxy. For further configuration, you can use the Caching Proxy configuration wizard, the Web-based administration tool or you can also edit the configuration file manually.

5.7.1 Using the Caching Proxy configuration wizard

The Caching Proxy configuration wizard allows you to easily create a reverse proxy configuration for your server and to add an administrator user.

To run the wizard on Windows systems click **Start -> Programs -> IBM WebSphere -> Edge Components -> Caching Proxy -> Configuration Wizard**. The configuration wizard requires that JRE 1.4.2 is installed on your system and in the system path.

To run the wizard on UNIX systems, run the command:

```
/opt/ibm/edge/cp/cpwizard/cpwizard.sh
```

1. Click **Next** on the welcome window. The following window allows you to configure the port that Caching Proxy will listen on for requests. The default value is 80 as shown in Figure 5-74. Enter the port number into the Port field and click **Next** to continue.

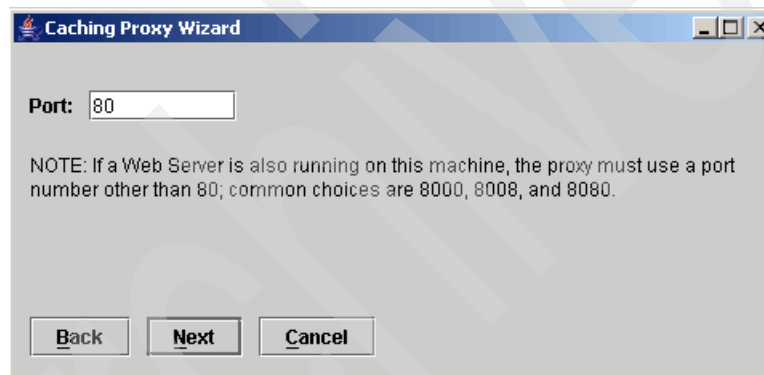


Figure 5-74 Setting the port number

Notes:

- ▶ The default port used by Caching Proxy is 80, which is also the default port of most Web servers. If you are collocating Caching Proxy and the Web server, you have two options regarding Caching Proxy:

- Use another port (for example, 81)
- Configure Caching Proxy to bind to a specific IP address

If you choose to configure Caching Proxy to bind to a specific IP address you need to make sure that the Web server is not binding to that IP address also. Refer to 5.7.3, “Manual configuration” on page 218 for an example of how to configure Caching Proxy to bind to a specific IP address.

- ▶ The default administration port used by Caching Proxy is 8008 which is identical with the IBM HTTP Server administration port. Therefore, if you collocate Caching Proxy with IBM HTTP server (or another Web server using the same administration port number), then you need to change the administration port directive `AdminPort` in the Caching Proxy configuration file `ibmproxy.conf` to another port number.

2. In the following window you need to enter the target Web server for the reverse Caching Proxy. In our scenario, Caching Proxy redirects the requests to the cluster called `cluster.itso.ibm.com`, which is handled by Dispatcher.

Note: If you are not using a cluster scenario, enter the hostname of the Web server to which the requests will be forwarded from the Caching Proxy.

Type the hostname of the Web server in the Target Web Server field and click **Next**, as shown in Figure 5-75.

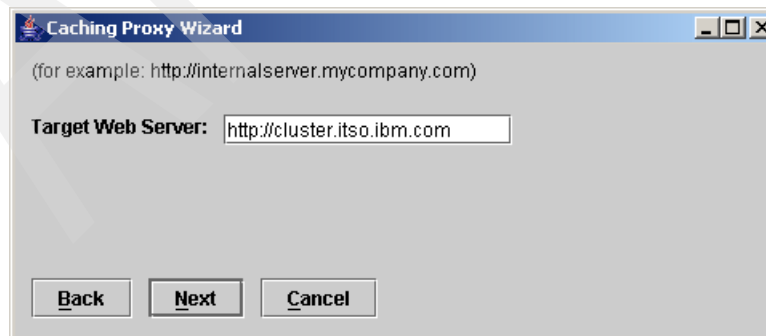
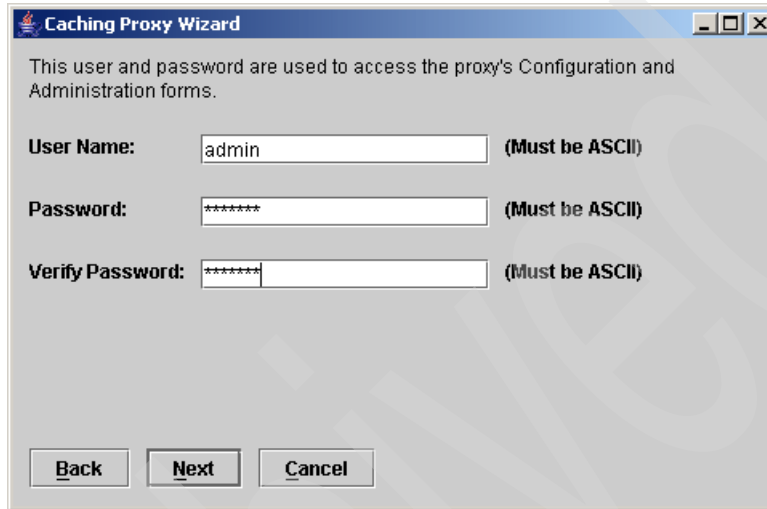


Figure 5-75 Setting the target Web server

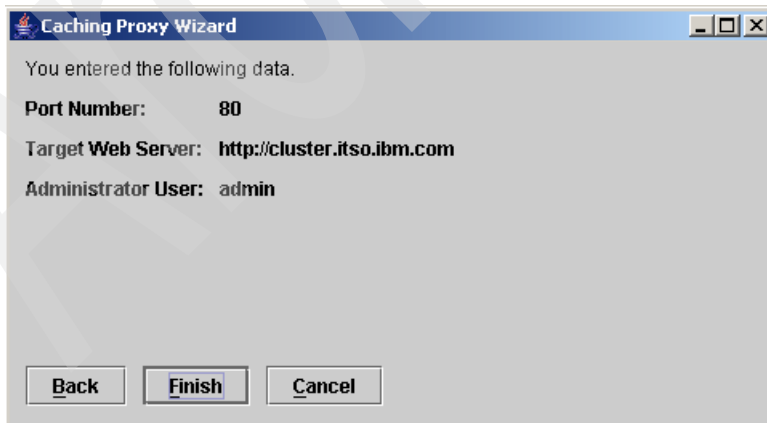
3. The following window allows you to create an administrator user for the Administration and Configuration forms (see 5.7.2, “Using the Caching Proxy Web-based administration tool” on page 216). Type the user name you want to create into the User Name field, type the password into the Password and Verify Password fields, and click **Next** as shown in Figure 5-76.



The screenshot shows a Windows-style dialog box titled "Caching Proxy Wizard". It contains a message: "This user and password are used to access the proxy's Configuration and Administration forms." Below this are three input fields: "User Name:" with the text "admin", "Password:" with masked characters "*****", and "Verify Password:" with masked characters "*****". Each field has a note "(Must be ASCII)" to its right. At the bottom are three buttons: "Back", "Next", and "Cancel".

Figure 5-76 Adding the administrator user

4. The last window shows a summary of the information you provided in the previous windows. Make sure everything is correct, and click **Finish** to apply the changes to the Caching Proxy configuration, as shown in Figure 5-77.



The screenshot shows a Windows-style dialog box titled "Caching Proxy Wizard". It contains a message: "You entered the following data." Below this are four lines of configuration data: "Port Number: 80", "Target Web Server: http://cluster.itso.ibm.com", and "Administrator User: admin". At the bottom are three buttons: "Back", "Finish", and "Cancel".

Figure 5-77 Summary of the configuration information

After the wizard is closed, you need to stop and start the Caching Proxy process for the changes to take effect. Refer to 5.8, “Managing the Caching Proxy process” on page 222.

5.7.2 Using the Caching Proxy Web-based administration tool

An administrator user ID and password needs to be created in order to administer the Caching Proxy through a browser. We have created the user when configuring the Caching Proxy as described in “Using the Caching Proxy configuration wizard” on page 213. If you have not created the user at that time, you can use the command line interface to do so or run the configuration wizard again. The commands are:

On UNIX:

```
htadm -adduser /opt/ibm/edge/cp/server_root/protect/webadmin.passwd
```

On Windows:

```
cd c:\Program Files\IBM\edge\cp\server_root\protect\  
htadm -adduser webadmin.passwd
```

When prompted, provide a username, password and name for the administrator.

After the Caching Proxy is started, you can then access the Front Page by typing the URL `http://<hostname>:<admin_port>/` in a supported browser, as shown in Figure 5-78 on page 217.

You can access the Caching Proxy administration tool by clicking the **Configuration and Administration Forms** link on the Front Page, or you can access it directly using the following URL:

```
http://<hostname>:<admin_port>/admin-bin/webexec/wte.html
```

Important: If you enable reverse proxy, the Front Page URL will no longer work. In order to access the Configuration and Administration Forms you need to use the direct URL.

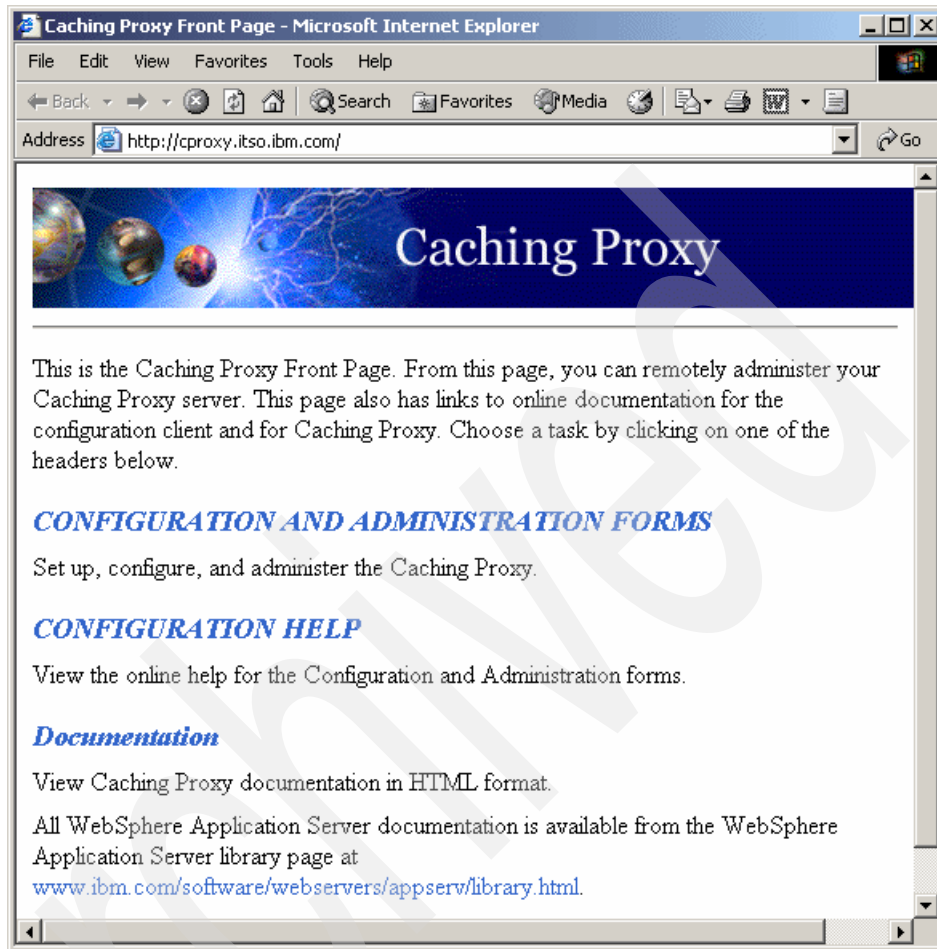


Figure 5-78 Caching Proxy front page

After clicking Configuration and Administration Forms, or accessing the direct URL, you need to provide the username and password you added previously.



Figure 5-79 Caching Proxy configuration and administration tool

5.7.3 Manual configuration

You can change Caching Proxy settings by editing the file `ibmproxy.conf` located in the `<product_install_path>\etc\en_US` (in Windows systems) or `<product_install_path>/etc/en_US` (in UNIX systems).

In AIX systems, the installation wizard adds a link to the configuration file in the `/etc` directory. So you can use the link `/etc/ibmproxy.conf` whenever you need to edit the configuration file.

Any change made directly to the file requires that you stop and start the server (refer to 5.8, “Managing the Caching Proxy process” on page 222).

For example, if you want Caching Proxy to bind to a specific IP address, you can configure it by editing the `ibmproxy.conf` file:

1. Locate the `BindSpecific` directive, and change it to “on”:

```
BindSpecific on
```

2. Add a Hostname directive specifying the full qualified hostname that resolves to the IP address that you want Caching Proxy to bind to:

```
Hostname cproxy.itso.ibm.com
```

3. Save the file, stop and start the Caching Proxy process.

This configuration is necessary if you want to collocate Caching Proxy with another process that also uses port 80, or with Load Balancer.

5.7.4 Creating and defining a cache storage

Caching Proxy by default stores the cache contents in memory. The default memory cache size can (and should) be increased when a memory-only cache is desired. However, it is not recommended that you set a memory cache larger than 1600 MB. You can also format disk areas to be used by Caching Proxy.

Creating a cache storage on disk

Caching Proxy uses raw disk caching, which means that it reads and writes directly to a partition on the disk device and does not use the regular file/directory API for performing I/O. This requires that the disk is prepared using a formatting tool provided by the product: **htcformat**.

When a raw cache device is specified, the memory cache becomes a buffer to the disk cache. In other words when a disk device is defined, the memory cache does not fill up first and then the raw device is used. Instead, the memory cache acts as a buffer space for the disk cache. When a disk cache is used, it is recommended that the memory cache be set to 1-2% the size of disk cache space but no less than the default value.

In our scenario, we are running Caching Proxy on an AIX server. In AIX systems, the area that is used as cache storage is a raw device. It is created as a logical volume, but it has no file system associated with it.

Use the following command to create the logical volume:

```
mklv -y <logical_volume_name> <volume_group_name> <number_partitions>
```

We first created the logical volume using the following command:

```
mklv -y lvcache01 cachevg 64
```

The partition size used in our volume group cachevg is 16MB (you can check the logical partition size for a volume group in AIX by using the command **lsvg <volume_group_name>**), so this command creates a logical volume named /dev/lvcache01 in the volume group cachevg, and sets its size to 1GB (64 partitions of 16MB each one).

After creating the logical volume, we need to format it using the `htcformat` tool. Note that although we named the logical volume `lvcache01`, we need to refer to the raw device, so we need to use the name `rlvcache01` (this is an operating system characteristic). See the `htcformat` command we used in Example 5-34.

Example 5-34 Running the `htcformat` command

```
# htcformat /dev/rlvcache01
Are you sure you want to format /dev/rlvcache01? (y/n): y
Formatting device /dev/rlvcache01, block size 8192, 131072 blocks ... Done
```

Note: Refer to the *Caching Proxy Administration Guide Version 6.0*, GC31-6857 for more information about `htcformat` and how to use it in other supported operating systems.

The storage is ready to be used by Caching Proxy. The next thing we need to do is to configure Caching Proxy to use this new cache area.

Access the Configuration and Administration Forms, click **Cache Configuration** -> **Cache Settings**. Locate the Specify the device or devices to be used for cache storage: field. Select **Add** and type the name of the new cache storage (in our scenario it is `/dev/rlvcache01`), and click **Submit** as shown in Figure 5-80 on page 221.

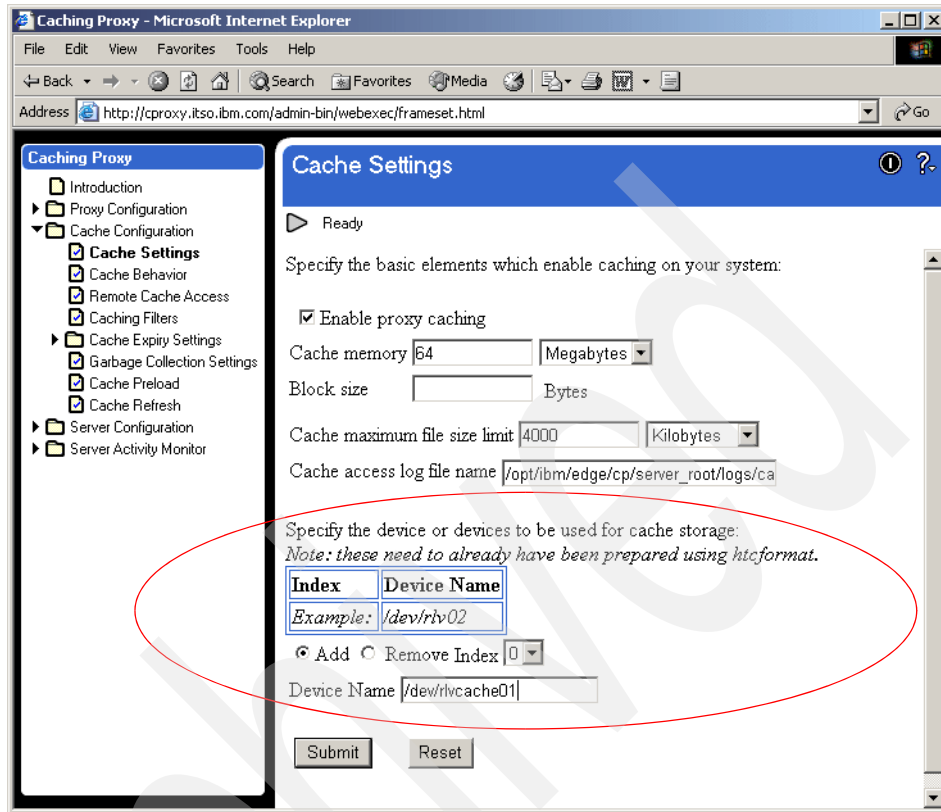


Figure 5-80 Adding a new cache storage

If you prefer to edit the configuration file instead of using the Configuration and Administration Forms, you can do it by adding the following line to the `ibmproxy.conf` file:

```
CacheDev                /dev/r1vcache01
```

After stopping and starting the server, you can check the log files to confirm that Caching Proxy was able to use the disk cache successfully.

Locate the event log file (by default, it is `/opt/ibm/edge/cp/server_root/logs/event.<timestamp>`) and check the messages in it. Example 5-35 shows the contents of our test server log file after we added the new disk cache storage.

Example 5-35 Event log entries

```
[21/Oct/2004:16:44:47 -0500] Edge Components Caching Proxy
[21/Oct/2004:16:44:47 -0500] Version: 6.0 .
[21/Oct/2004:16:44:47 -0500] Initializing disk cache.
```

```

[21/Oct/2004:16:44:47 -0500] block size is 8192 bytes
[21/Oct/2004:16:44:47 -0500] cache memory is 65536 KBytes
[21/Oct/2004:16:44:47 -0500] number of devices is 1
[21/Oct/2004:16:44:47 -0500] Cache device is /dev/rlvcache01
[21/Oct/2004:16:44:47 -0500] garbage collection policy is bandwidth
[21/Oct/2004:16:44:47 -0500] garbage collection low water mark is 60 percent
[21/Oct/2004:16:44:47 -0500] garbage collection high water mark is 90 percent
[21/Oct/2004:16:44:47 -0500] garbage collection startup callout is 0
[21/Oct/2004:16:44:47 -0500] garbage collection evaluation callout is 0
[21/Oct/2004:16:44:47 -0500] garbage collection shutdown callout is 0
[21/Oct/2004:16:44:47 -0500] garbage collection recent access is 60 seconds
[21/Oct/2004:16:44:47 -0500] garbage collection maximum lifetime is 2592000
seconds
[21/Oct/2004:16:44:47 -0500] garbage collection minimum size is 0 bytes
[21/Oct/2004:16:44:47 -0500] garbage collection maximum size is 0 bytes
[21/Oct/2004:16:44:47 -0500] garbage collection load threshold is 30 seconds
[21/Oct/2004:16:44:47 -0500] Disk cache is using 131072 blocks on 1 devices
[21/Oct/2004:16:44:47 -0500] container size is 1024 blocks
[21/Oct/2004:16:44:47 -0500] extent size is 8 blocks
[21/Oct/2004:16:44:47 -0500] header cache size is 13107 headers
[21/Oct/2004:16:44:47 -0500] header block cache size is 1024 blocks
[21/Oct/2004:16:44:47 -0500] data block cache size is 6730 blocks
[21/Oct/2004:16:44:47 -0500] file hash table has 11003 buckets
[21/Oct/2004:16:44:47 -0500] header hash table has 3001 buckets
[21/Oct/2004:16:44:47 -0500] block hash table has 2003 buckets
[21/Oct/2004:16:44:47 -0500] device hash table has 1009 buckets
[21/Oct/2004:16:44:47 -0500] number of temporary containers is 1024
[21/Oct/2004:16:44:47 -0500] number of writer threads is 1
[21/Oct/2004:16:44:47 -0500] number of compactor threads is 1
[21/Oct/2004:16:44:47 -0500] number of garbage collector threads is 1
[21/Oct/2004:16:44:47 -0500] cache write size is 8 extents
[21/Oct/2004:16:44:47 -0500] compactor read/write size is 8 extents
[21/Oct/2004:16:44:47 -0500] header write size is 7864 headers
[21/Oct/2004:16:44:47 -0500] Starting cache consistency check
[21/Oct/2004:16:44:47 -0500] Formatting /dev/rlvcache01
[21/Oct/2004:16:45:40 -0500] Finished formatting /dev/rlvcache01
[21/Oct/2004:16:45:40 -0500] Finished cache consistency check
[21/Oct/2004:16:45:40 -0500] The proxy cache is ready

```

Note: It may take a while (up to a few minutes) to init the disk cache device.

5.8 Managing the Caching Proxy process

In Windows systems you can start and stop the Caching Proxy process by using the Windows Services tool and locating the IBM Caching Proxy service, as shown in Figure 5-81 on page 223.

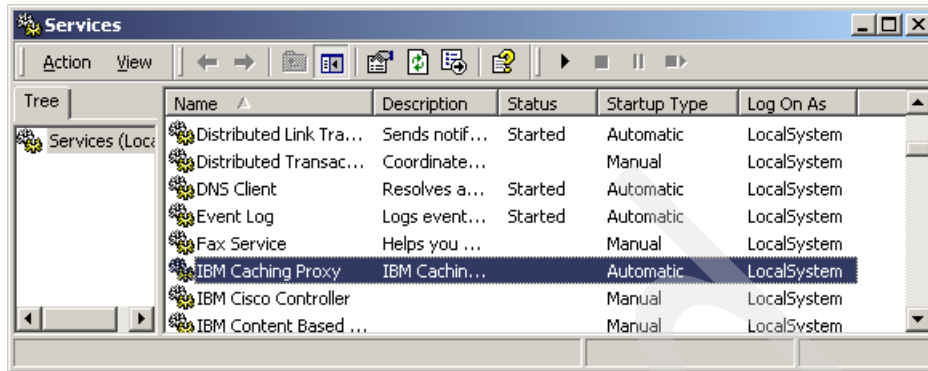


Figure 5-81 Caching Proxy service

In AIX systems, you can use the System Resource Controller commands to administer the ibmproxy daemon. The commands are:

- ▶ To start the daemon:


```
startsrc -s ibmproxy
```
- ▶ To stop the daemon:


```
stopsrc -s ibmproxy
```
- ▶ To check the status of the daemon:


```
lssrc -s ibmproxy
```
- ▶ To stop the daemon on other UNIX operating systems:


```
Kill -TERM ibmproxy-pid
```

Tip: It is recommended to stop the Caching Proxy using the appropriate command for each operating system. If the Caching Proxy is not shutdown properly, it might take a long time to rescan/verify disk cache integrity.

If you are using the Configuration and Administration Forms, you can restart the server by clicking the Restart button in the upper right corner of the window, as shown in Figure 5-82.



Figure 5-82 Restart button

Important: Some changes require you to stop and start the Caching Proxy process; a restart is not enough. Refer to “Directives not changed on restart” in Appendix B of *Caching Proxy Administration Guide Version 6.0*, GC31-6857 for more information about which changes require a stop and start of the Caching Proxy process.

5.8.1 Testing the Caching Proxy scenario

At this point, we have the Load Balancer servers up and running as described in 5.2, “Load Balancer configuration: basic scenario” on page 135 or 5.3, “Load Balancer: high availability scenario” on page 162.

We configured Caching Proxy to act as a reverse proxy for our Load Balancer cluster (see 5.7.1, “Using the Caching Proxy configuration wizard” on page 213). This means that all requests we send to the Caching Proxy will be forwarded to the Load Balancer cluster address, and subsequently they are balanced between our two Web servers (http1 and http2). Finally the request is handled by the application servers in the cluster (refer to Chapter 8, “Implementing the sample topology” on page 387 for more information about our test scenario).

In order to test your configuration, open a browser and request the BeenThere application URI using the hostname of the Caching Proxy server:

```
http://cproxy.itso.ibm.com/wlm/BeenThere
```

Note: If your Caching Proxy is collocated with a Web Server, do not forget to use the correct port number for the request, for example:

```
http://cproxy.itso.ibm.com:81/wlm/BeenThere
```

After generating several requests, access the Administration and Configuration Forms of Caching Proxy to check the cache usage.

Log in, click **Server Activity Monitor -> Proxy Access Statistics**. This pane shows the requests received by Caching Proxy, the lines in blue are requests that were delivered from the local cache, the ones in black are requests that were sent to the back-end servers.

In our test, Caching Proxy cached the file /wlm/success_banner.jpg and only forwarded the requests for the servlet to the back-end servers, as shown in Figure 5-83 on page 225.

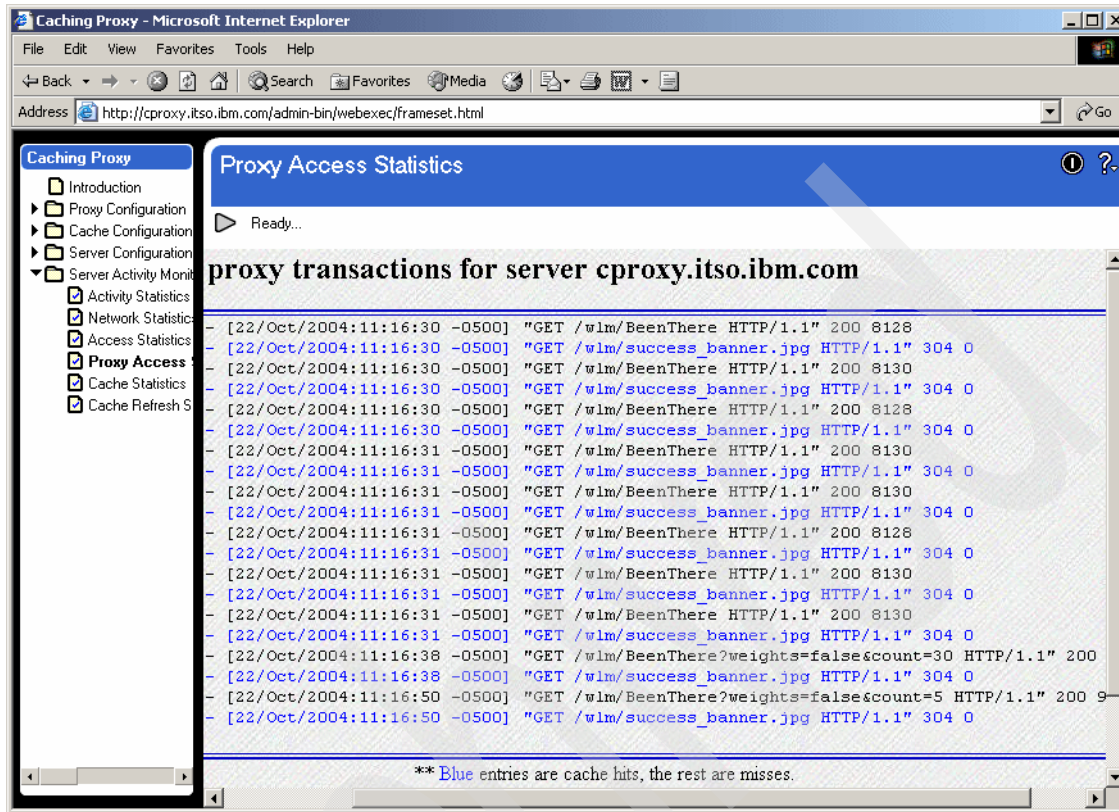


Figure 5-83 Cache statistics for BeenThere

We also tested the Trade Application by requesting the URL

<http://cproxy.itso.ibm.com:<port>/trade>

Looking at the Proxy Access Statistics, we can see that Trade has more static contents than BeenThere, so the number of cache hits is higher when compared to BeenThere, as shown in Figure 5-84 on page 226.

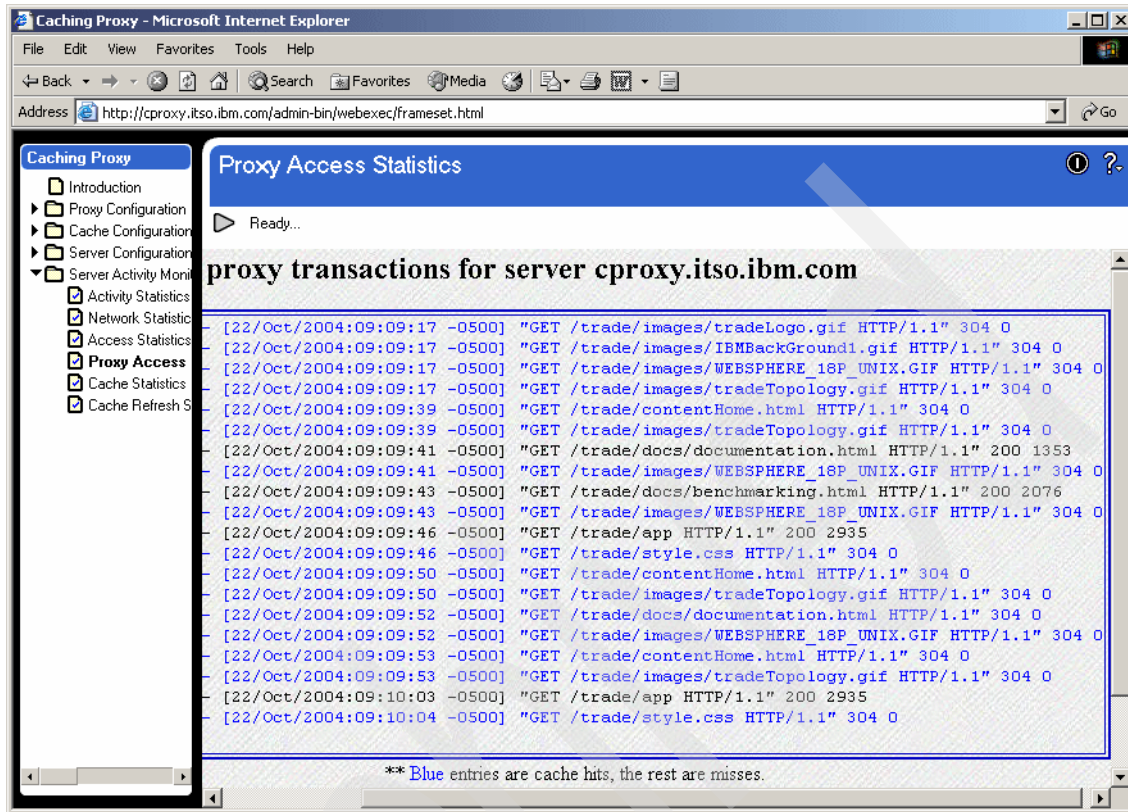


Figure 5-84 Cache statistics for Trade

Note that we are not using dynamic caching in this scenario. For more information about dynamic caching refer to Chapter 10, "Dynamic caching" on page 501.

Plug-in workload management and failover

In this chapter, we cover how to set up the Web container for workload management. We discuss the components that make this possible, the configuration settings, and the resulting behaviors. We also cover session management in a workload-managed environment and the process of troubleshooting all these areas.

This chapter covers in detail:

- ▶ WebContainer transport chains and virtual hosts
- ▶ Web server topologies
- ▶ WebSphere plug-in configuration file
- ▶ WebSphere plug-in workload management
- ▶ Web container failures and failover
- ▶ HTTP session management
- ▶ Troubleshooting the Web server plug-in
- ▶ WebSphere plug-in behavior

6.1 Introduction

The Web container manages the J2EE components, servlets and JSPs that are accessed from the Web.

Traffic for these components is workload-managed by configuring clusters within a cell in WebSphere Application Server and then configuring Web servers to access cluster members.

WebSphere Application Server V6 supports a variety of Web servers. These Web servers are necessary for directing traffic from users' browsers to the applications running in WebSphere. This is achieved by installing the WebSphere plug-in on the Web servers. As shown in Figure 6-1 on page 229, the plug-in is a module that runs as part of the Web server process. It acts as a router, deciding what traffic is for WebSphere and what traffic should be handled by the Web server itself.

This workload management process also sets up failover and backup servers in the event that one or more of the cluster members should fail or all primary servers fail.

The Web containers may be running on the same machine as the Web server, a different machine, or a combination of the two.

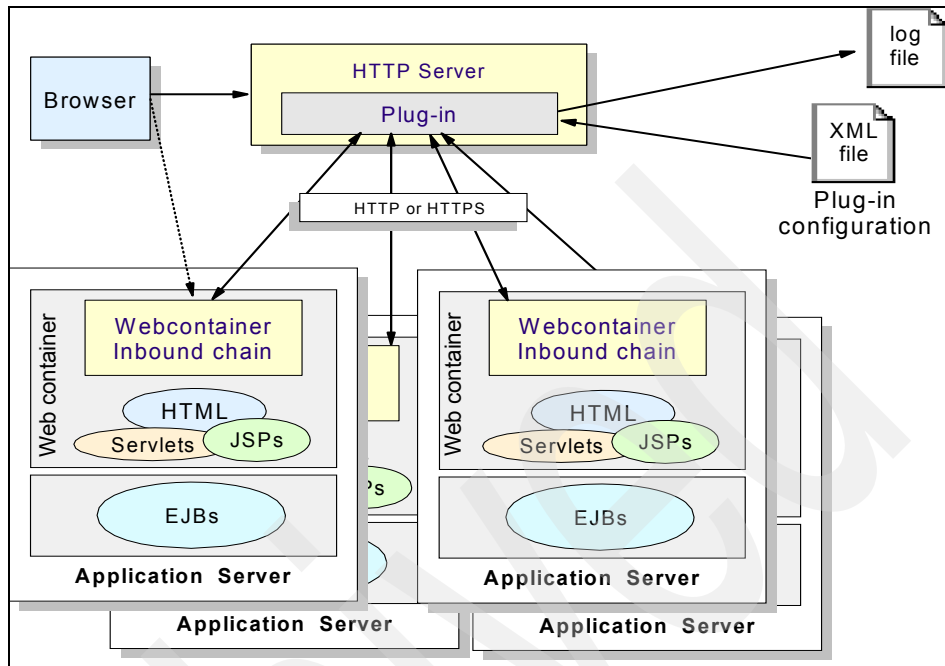


Figure 6-1 Plug-in components and interactions

The Web server plug-in uses HTTP and HTTPS as its transport protocol between the Web server and the WebSphere Application Server(s). The WebSphere plug-in was introduced in WebSphere 4.0. Since then, the WebSphere Application Server plug-in gained some new features, such as:

- ▶ Weighted round robin and random workload management
- ▶ Backup servers

In this chapter we are using a subset of the sample topology described in Figure 8-1 on page 390 to demonstrate plug-in work load management. This subset is illustrated in Figure 6-2 on page 230.

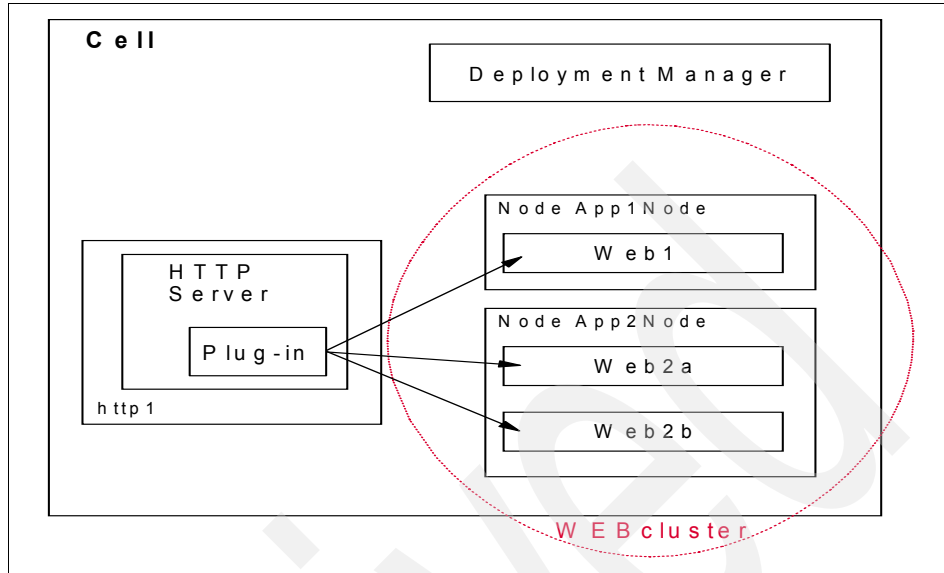


Figure 6-2 Topology used in this chapter

A new feature in WebSphere Application Server V6 is that Web servers are now considered part of the cell. Naturally, Web servers were always part of the topology managed by administrators — the new feature is that now they are explicitly mentioned in the Administrative Console, with some very interesting consequences regarding the plug-in configuration file management (especially for those who use the IBM HTTP Server V6.0). You will find more information about this topic in 6.4, “Web server topologies” on page 244 and 6.6, “WebSphere plug-in workload management” on page 264.

6.2 WebContainer transport chains and virtual hosts

The workload management performed by the plug-in distributes the load between available Web containers. Thus, the Web containers need to be set up correctly so that the plug-in can route traffic to them.

This chapter discusses setting up transport chains, virtual hosts, Web servers, clusters, and cluster members. By understanding these you will learn how and over what connection the plug-in is routing requests.

6.2.1 WebContainer Inbound Chains

Before moving on to the main part of this chapter, it is important to discuss and understand a new function in WebSphere Application Server V6 called *transport chains*. WebSphere V6 introduced transport chains as a new communications implementation that, although unnoticed most of the time, changes the way communication through protocol stacks works.

As in earlier WebSphere versions, it is still true that each application server has a built-in HTTP server, which is bound to Web container transport chains. In earlier versions of WebSphere, we called this built-in HTTP server the “embedded HTTP transport”, from now on, we refer to it as the *WebContainer Inbound Chain*. One advantage of this new implementation is that it offers a much higher request throughput.

The HTTP service is bound to the transport chain named `WCInboundDefault`, and the HTTPS service is bound to the one named `WCInboundDefaultSecure`.

The transport chains of a Web container can be found and manipulated through the Administrative Console by selecting **Servers -> Application servers -> <AppServer_Name> -> Web container transport chains**, as shown in Figure 6-4 on page 235.

Using the internal transport in the Web container is extremely useful for testing and in development environments because there is no need to configure a separate Web server or to update the Web server plug-in when changes are made to URL mappings. A user can establish a direct connection from the browser to the application server without the need for a separate Web server, just using a URL with the application server host name and the `WCInboundDefault` transport number:

```
http://app1.itso.ibm.com:9080/snoop
```

However, it is strongly recommended that this internal Web server should not be used in production environments. If the users connect directly to the *WebContainer Inbound Chain*, then they bypass the plug-in and the workload management it performs, as well as the failover and session affinity mechanisms it provides — not to mention the ESI dynamic caching function in the Web server plug-in, if enabled.

Further information about this is found in 6.2.3, “Transport chains: the details” on page 234.

6.2.2 Virtual hosts

The Web containers handle traffic for the Web modules contained within them. WebSphere uses *virtual hosts* as filters for the traffic coming in from the Web. A virtual host is a configuration enabling a single host machine to resemble multiple host machines.

Each virtual host has a logical name and a list of one or more DNS aliases by which it is known. A DNS alias is the TCP/IP host name and port number used to request the servlet, for example yourHostName:80. When no port number is specified, 80 is assumed as the default.

When a servlet request is made, the server name and port number entered into the browser are compared to a list of all known aliases in an effort to locate the correct virtual host and serve the servlet. If no match is found, an error is returned to the browser.

A virtual host is not associated with a particular node (machine). It is a configuration, rather than a “live object,” explaining why it can be created, but not started or stopped.

For many users, virtual host creation is unnecessary because, by default, WebSphere Application Server V6 provides two virtual hosts:

- ▶ The default_host, which is used for accessing most applications.
- ▶ The admin_host, which is configured for accessing the Administrative Console (while other applications are not accessible through this virtual host).

The default settings for default_host map to all requests for any alias on ports 80, 9443, and 9080. The default settings for admin_host map to requests on ports 9060 and 9043.

The virtual host definition is a list of hosts and ports on which a Web module accepts traffic. As shown in Figure 6-3 on page 233, the default_host virtual host uses a wildcard (*) to allow traffic from all hosts on ports 80, 9080 and 9443. The settings for virtual host aliases can be found in the Administrative Console by clicking **Environment -> Virtual Hosts -> <Virtual_Host_Name> -> Host Aliases**.

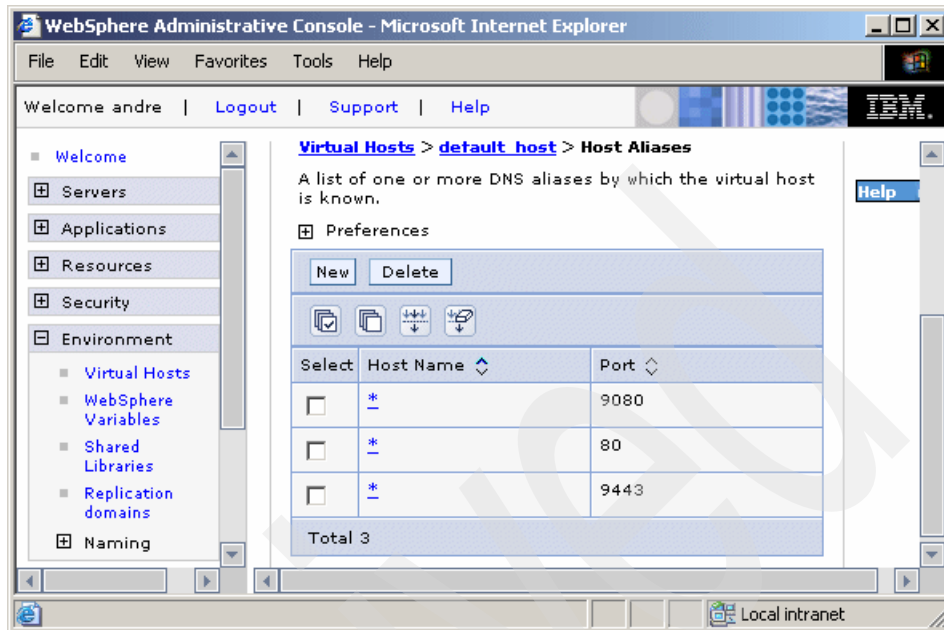


Figure 6-3 Virtual host setup

Tip: We recommend adding wildcard settings for ports 80, 9080, and 9443 by default for new virtual hosts.

However, with such a configuration, any Web server can redirect traffic to your Web containers. Use specific host names instead of wildcards on port 80 if you wish to prevent any Web server from redirecting traffic to your Web containers.

Virtual hosts are especially useful when setting up multiple application servers that use the same URI. Virtual host filtering can be used so that the requests `www.vhost1.com/snoop` and `www.vhost2.com/snoop` can be handled by two different Web modules. In such a case, you naturally use the specific host names, not wildcards, when defining the virtual hosts. Follow these steps to set this up:

1. Define two virtual hosts (for example, `vh1` and `vh2`).
2. Configure the `www.vhost1.com` host alias in `vh1` (port 80) and `www.vhost2.com` in `vh2` (also in port 80).

For more information about configuring virtual hosts, see section 5.7, “Working with virtual hosts” of the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

Note: You may notice that in WebSphere V6 the plug-in is more forgiving when the same wildcard aliases exist in several virtual hosts.

Still, some configurations might lead to a situation where the plug-in and the Web container are not able to correctly match a virtual host with the request URL received from the Web client (this happens when the associated host and port do not exist in the virtual host group).

The error that would be seen would be similar to this:

```
PLGN0021E: Servlet Request Processor Exception: Virtual Host/WebGroup Not  
Found : The web group /xx has not been defined
```

This is a generic error message and it can mean a number of things. You could check whether the Web module is mapped to the correct virtual host or if the plug-in configuration has been updated and reloaded by the Web server. If you are sure that everything is configured correctly, then you should use specific host names on the host aliases entries.

6.2.3 Transport chains: the details

Each application server has a Web container and transports defined for it. This information can be found by clicking **Servers -> Application servers -> <AppServer_Name> -> Web Container Settings -> Web container transport chains** as shown in Figure 6-4 on page 235. As can be seen, we have several transport chains configured for a typical Web container; in our example, port 9081 is being used for normal HTTP communication.

Note: As you can see in Figure 6-4, there are also two administration-related transport chains in our default configuration. These transport chains will actually not be used by the plug-in. In fact, when the plug-in file is generated, the generator skips all transport chains that are related to administration.

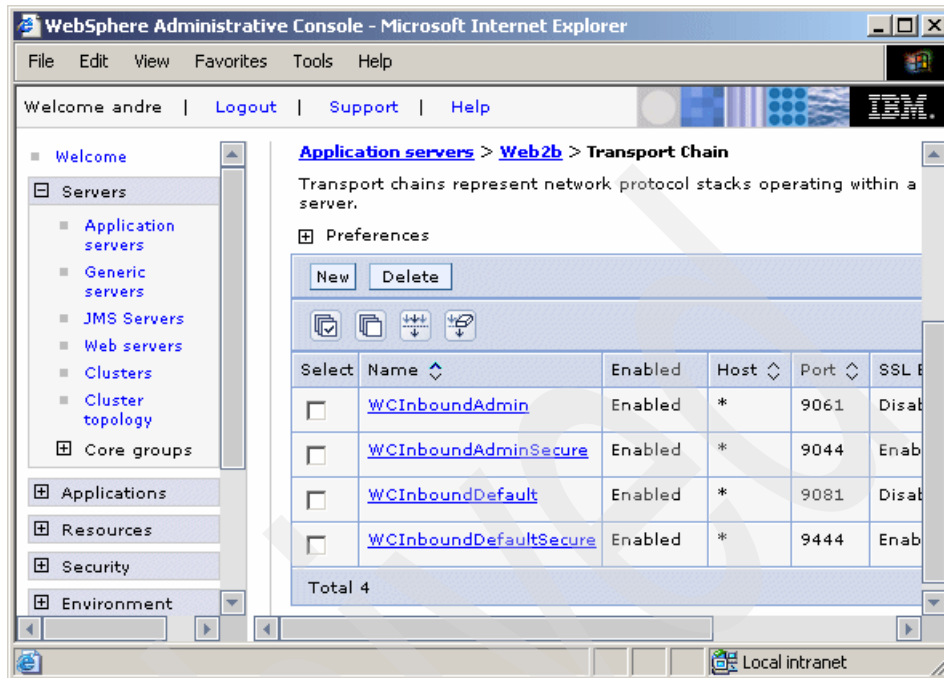


Figure 6-4 Typical transport settings for Web container (ports may vary)

As discussed previously, a transport chain defines a communication mechanism bound to a TCP/IP port that is used by the plug-in (or any HTTP client) to access a Web container. The transport chain associated with each Web container contains an HTTP channel that handles HTTP requests forwarded from the Web server plug-in.

It is important to understand the transport chains, since their settings directly affect how the plug-in works. Also, by understanding and setting up transports as the first part of a cluster member setup, time can be saved. This is because, in a workload-managed environment, transport settings are unique to each node. When a cluster member is added, you have the option to generate unique port numbers for the cluster member as shown in Figure 6-5 on page 236. WebSphere Application Server then allocates unique port numbers and defines the necessary transport settings for each application server.

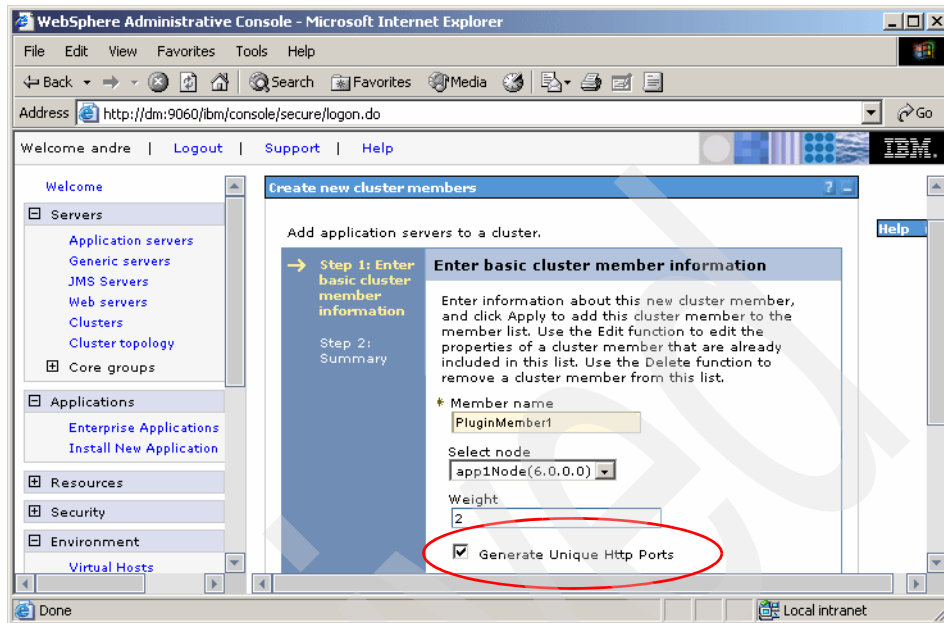


Figure 6-5 Generating unique port numbers

Setting up transport chains

A newly created application server will typically have four transport chains defined, as detailed in Table 6-1:

Table 6-1 Typical Web container transport chains

Transport chain	Purpose
WCInboundAdmin	On a stand-alone server, it is used by the admin_host virtual host, which hosts the WebSphere Administrative Console.
WCInboundAdminSecure	Same function as WCInboundAdmin, but for HTTPS communication.
WCInboundDefault	This transport chain is connected to the default HTTP channel associated with the Web container. You can use this transport chain directly to bypass the Web server plug-in.
WCInboundDefaultSecure	Same function as WCInboundDefault, but for HTTPS communication.

To inspect a transport chain more closely, click the appropriate link (WCInboundDefault, for example). This brings up a window like the one shown in Figure 6-6:

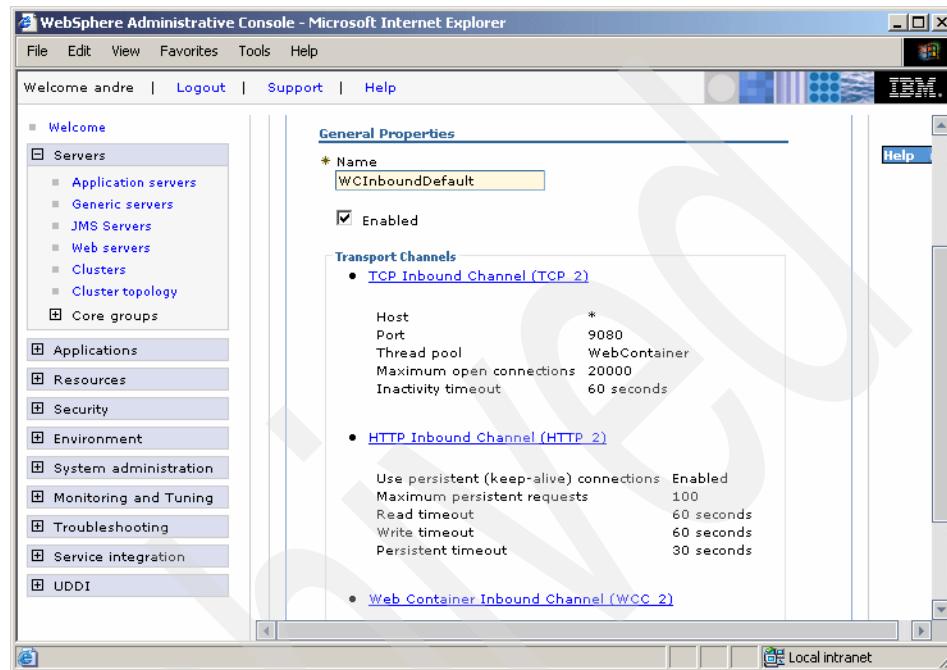


Figure 6-6 A transport chain and its channels

As you can see here, a transport chain contains several *Transport Channels* that reflect the protocol stack involved in the communication, such as TCP or HTTP. Now you can click, for example, the TCP channel link to configure it further for your particular needs, as shown in Figure 6-7 on page 238.

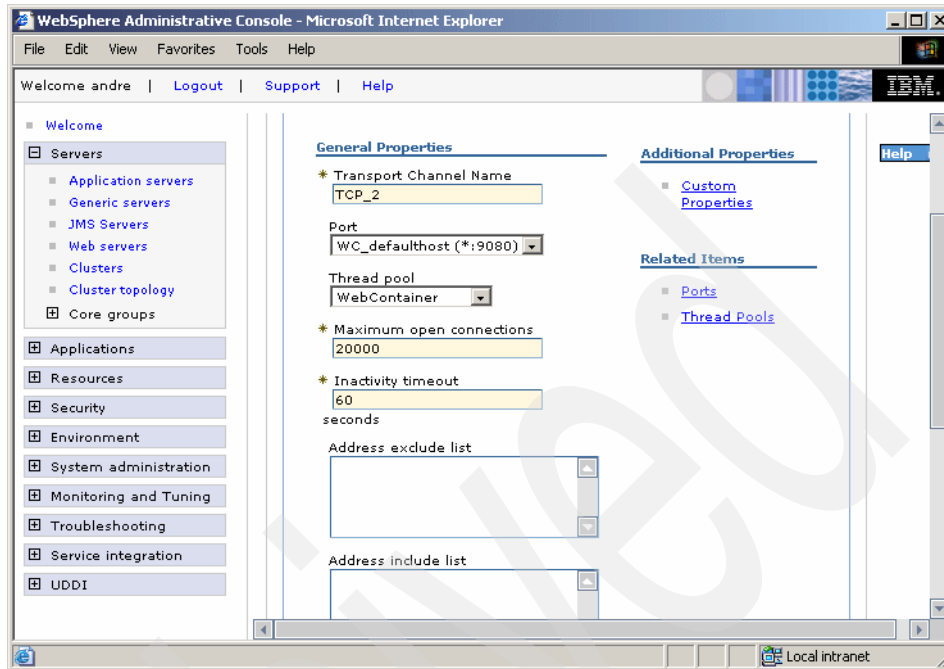


Figure 6-7 TCP channel settings

Ports

By default, individual ports are created automatically for each transport whenever a new application server is created. WebSphere Application Server is smart enough to generate port values that do not conflict with other WebSphere servers on the same node, but the administrator should be careful not to generate conflicts with other services. Port values can always be changed later should port conflicts ever show up in the error log.

The port associated with the TCP channel in our example can be defined by clicking the **Ports** link in the **Related Items** section. Next, select the respective port name link to display the configuration settings as shown in Figure 6-8 on page 239 (another way to do this is by selecting **Servers** -> **Application servers** -> **<AppServer_Name>** -> **Ports** -> **<Port_Name>**).

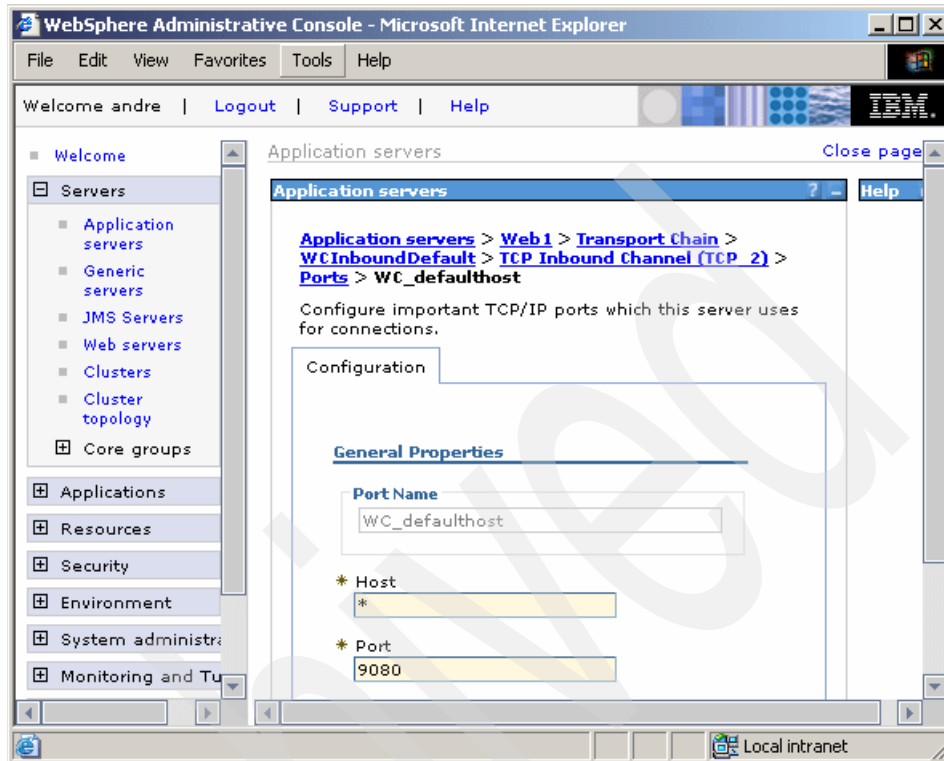


Figure 6-8 TCP channel port

The Host is the host IP address to which the plug-in will attempt to connect. If “*” is used, then WebSphere Application Server replaces the * with the application server host name when the plug-in configuration is regenerated (see 6.5.2, “Generation of the plug-in configuration file” on page 258). This is the recommended method when setting up transports in a multi-cluster member, multi-machine environment.

If you wish to use the secure transport chains, refer to Chapter 6, “Securing a Web application”, of the redbook *WebSphere Application Server V6: Security Handbook*, SG24-6316, for a more detailed look at SSL configuration.

Setting up multiple transport chains

If you define multiple transports chains of the same type (for example, HTTP) for one application server, then the plug-in will try to cross-reference a port number specified in the VirtualHostGroup to a transport specified for a plug-in member.

Setting up multiple transport chains requires the following steps:

1. Create new port(s)
2. Create new transport chain(s)
3. Add port numbers to list of host aliases

New ports are created by clicking **Servers -> Application servers -> <AppServer_Name> -> Ports -> New** as shown in Figure 6-9:

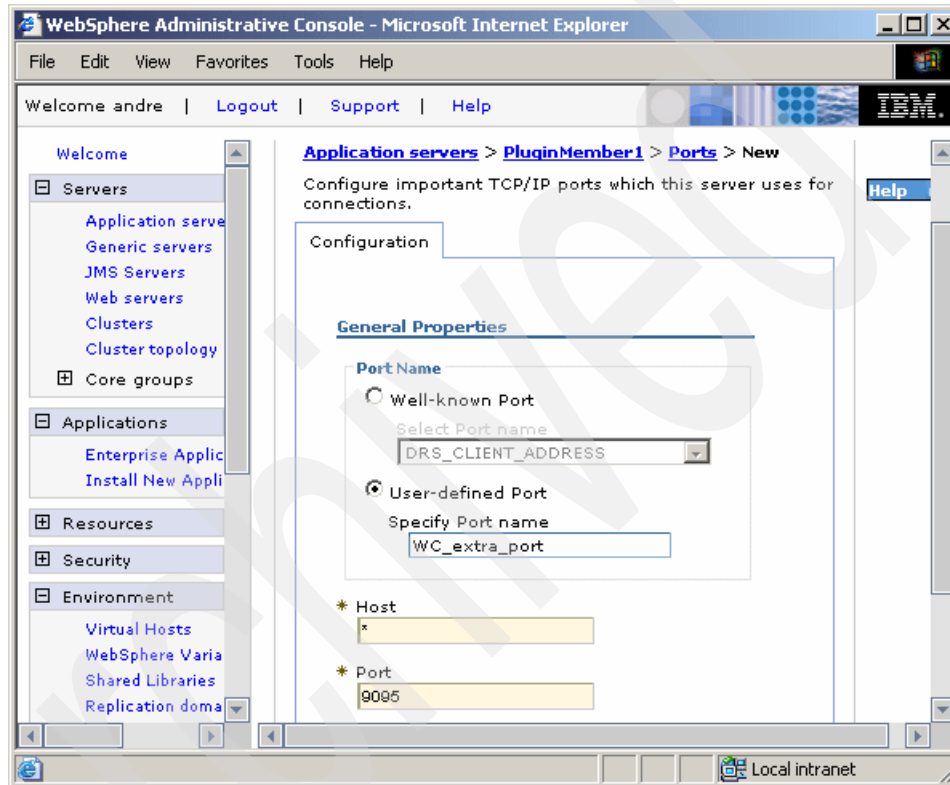


Figure 6-9 Configuring a new port

After configuring a new port for a server, you need to assign it to a new transport chain by clicking **Servers -> Application servers -> <AppServer_Name> -> Web Container Settings -> Web container transport chains -> New** (as can be seen in Figure 6-10 on page 241 and in Figure 6-11 on page 242).

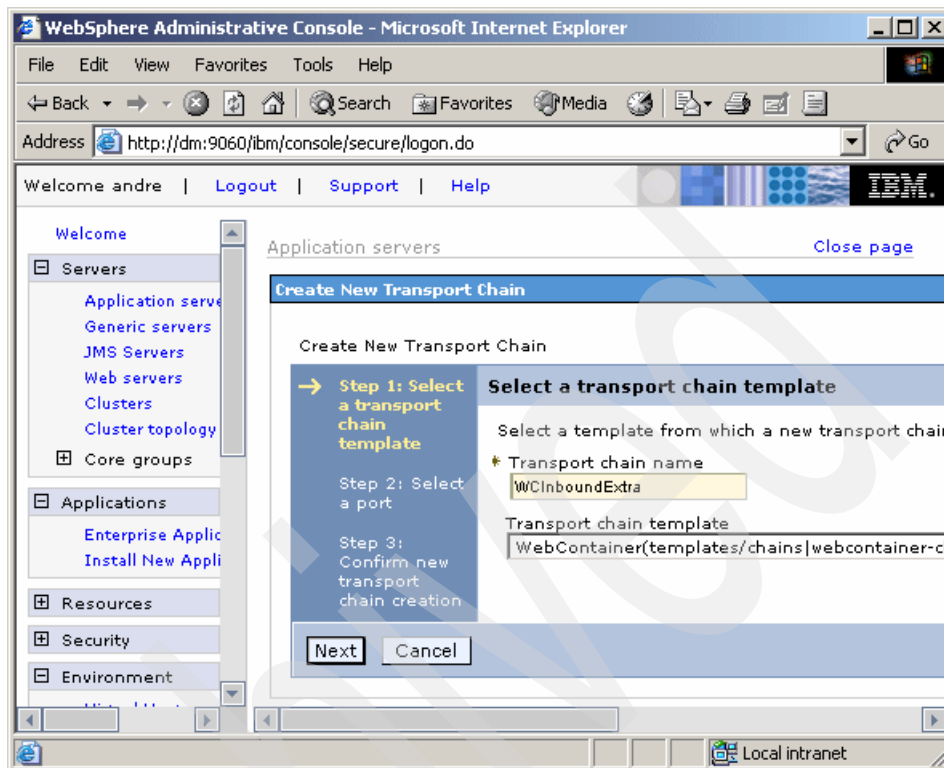


Figure 6-10 Configuring multiple transports

On the “Step 1” panel, you can enter the name of your new transport chain. Notice that an appropriate transport chain template must be chosen (WebContainer or WebContainer-Secure) before you click **Next**.

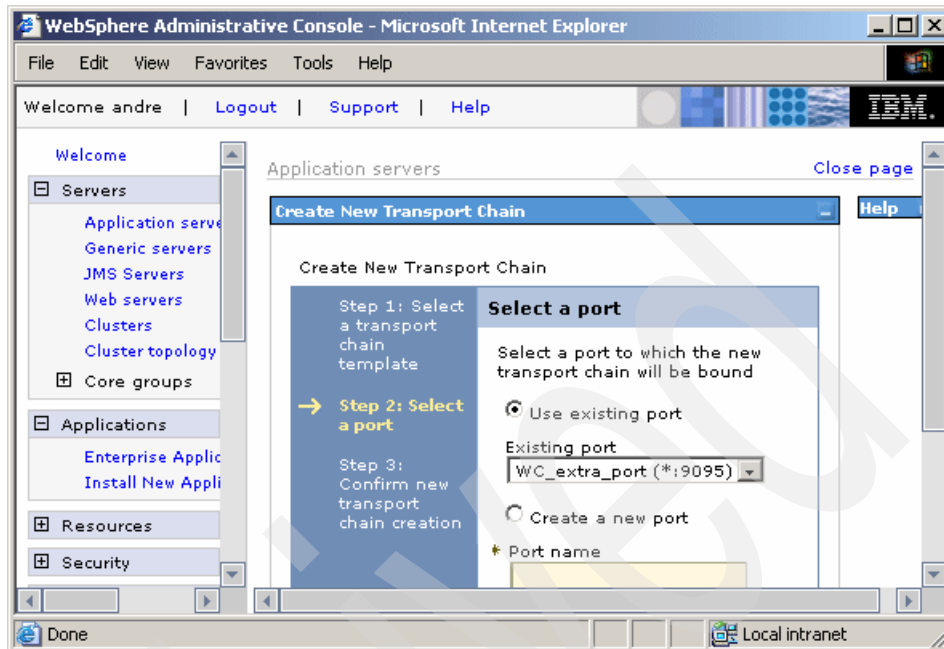


Figure 6-11 Assigning a port to the new transport

On the next panel, you can either select a port you previously created from the **Existing port** pull-down menu or you can create a new port on the fly. Click **Next** and then **Finish**. Do not forget to save your changes.

If you want this transport chain to be usable from a Web client, then the last configuration step is to add the port number to the list of host aliases of a virtual host, as explained in 6.2.2, “Virtual hosts” on page 232.

An example of the type of plug-in configuration generated by such a configuration is shown in Example 6-1 on page 243.

Important: All plugin-cfg.xml extracts shown in this book are for educational purposes only. Usually, you do not need to manipulate these files directly. Since WebSphere Application Server V6, most changes (with only a very few exceptions) can and should be done via the Administrative Console.

One exception is the ClusterAddress tag, which cannot be set using the Administrative Console. Please refer to “ClusterAddress” on page 271 for information about this setting.

```
...
<VirtualHostGroup Name="default_host">
  <VirtualHost Name="*:9080"/>
  <VirtualHost Name="*:80"/>
  <VirtualHost Name="*:9443"/>
  <VirtualHost Name="*:9081"/>
  <VirtualHost Name="*:9082"/>
</VirtualHostGroup>
  <ServerCluster CloneSeparatorChange="false" LoadBalance="Round Robin"
Name="WEBcluster" PostSizeLimit="-1" RemoveSpecialHeaders="true"
RetryInterval="120">
  <Server CloneID="vve2m4fh" ConnectTimeout="5" ExtendedHandshake="false"
LoadBalanceWeight="2" MaxConnections="-1" Name="app1Node_Web1"
WaitForContinue="false">
    <Transport Hostname="app1.itso.ibm.com" Port="9080" Protocol="http"/>
    <Transport Hostname="app1.itso.ibm.com" Port="9095" Protocol="http"/>
  </Server>
  .....
  <PrimaryServers>
    <Server Name="app2Node_Web2a"/>
    <Server Name="app2Node_Web2b"/>
    <Server Name="app1Node_Web1"/>
  </PrimaryServers>
</ServerCluster>
  <UriGroup Name="default_host_WEBcluster_URIs">
    <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid"
Name="/trade/*"/>
    <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid"
Name="/wlm/*"/>
  </UriGroup>
  <Route ServerCluster="WEBcluster" UriGroup="default_host_WEBcluster_URIs"
VirtualHostGroup="default_host"/>
...

```

Using Example 6-1, if a request to `http://Webserver/trade` was sent to the plug-in and the server Web1 was chosen for the request, the request would be sent to `app1.itso.ibm.com` on port 9080. This is because the plug-in has cross-referenced the ports specified in the transports of Web1 and the ports specified in the VirtualHostGroup `default_host`. The match between the two is port 9080; hence the request is sent to the Web container transport chain on port 9080. If no match is found, then the last transport chain for Web1 is used, in this case port 9095.

Note: Adding additional transport definitions will not increase the load balancing or failover capabilities of an application server.

HTTPS considerations in a WLM environment

When using secure Web container transport chains with a self-signed certificate, you need to distribute client (plug-in) and server (application server) certificates as follows:

- ▶ Import the server certificate from each application server as a trusted CA (certificate authority) into the plug-in's keyfile.
- ▶ If you are using client (plug-in) authentication, you also need to import the client certificate from each plug-in as a trusted CA (certificate authority) into the application server's keyfile.

For details about WebSphere Security, refer to the Redpaper *WebSphere security fundamentals*, REDP-3944 and to the redbook *WebSphere Application Server V6: Security Handbook*, SG24-6316.

6.3 Creating clusters and cluster members

To allow for workload management and failover, replicas of a Web module are created. This is done by creating more servers as cluster members. The concepts of clusters and cluster members are covered in 1.3.3, "Workload management using WebSphere clustering" on page 19. Information about how to set up clusters and cluster members can be found in 8.5, "Installing WebSphere and configuring clusters" on page 395, where the cluster configuration of our sample topology is done step by step.

Note: Clustering is performed at the application server level, not the Web module or EJB level.

6.4 Web server topologies

Since WebSphere Application Server V6, Web servers are an important conceptual part of a cell configuration. In some cases, it is now possible to *manage* a Web server from the WebSphere Administrative Console (you can stop and start its process, for example). It may also be possible to *propagate* a plug-in configuration file from the Administrative Console to the proper location in the remote Web server machine (see 6.5.3, "Propagation of the plug-in file" on page 262 for more information).

Whether or not you can manage a Web server (and propagate its plug-in configuration file) depends on the Web server type and its configuration in the cell. A configured Web server can fall into three categories:

- Managed Web servers
- Unmanaged Web servers
- IBM HTTP Server V6.0 (special case of unmanaged Web servers)

For details about Web server management in WebSphere V6, please refer to Chapter 8, “Managing Web servers” of the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

You should have the Web server already installed and running before configuring it in the cell. Information about how to install and configure the IBM HTTP Server V6.0 is found in 8.6, “Installing and configuring IBM HTTP Server 6.0” on page 416.

Web servers in a cell can be seen in the WebSphere Administrative Console by selecting **Servers -> Web servers**, as shown in Figure 6-12.

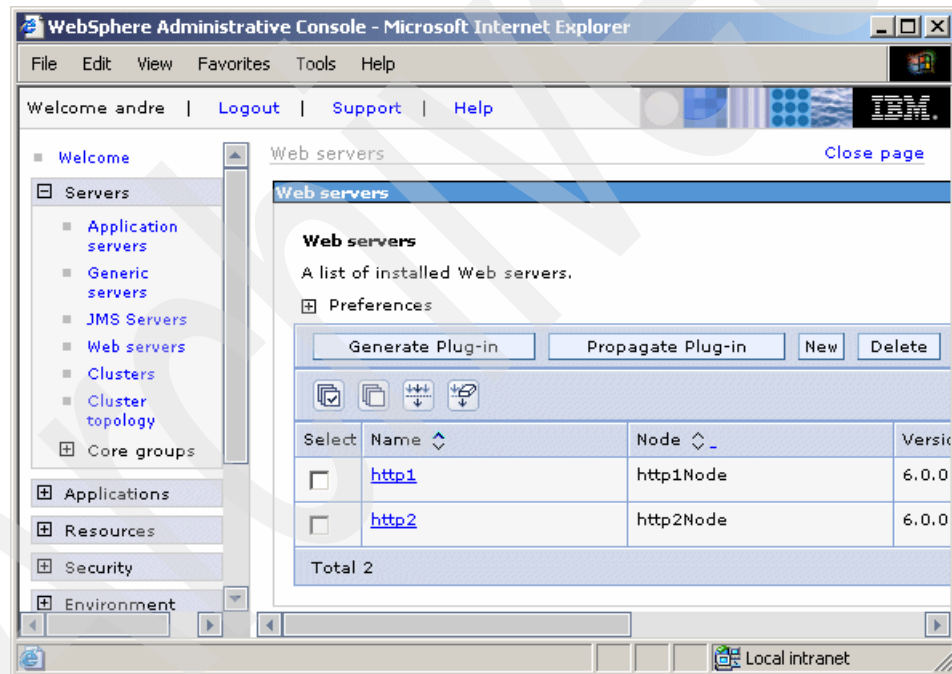


Figure 6-12 Web servers in a cell

In WebSphere V6, during application installation, you should map the Web modules to the appropriate Web servers, as shown in Figure 6-13 on page 246.

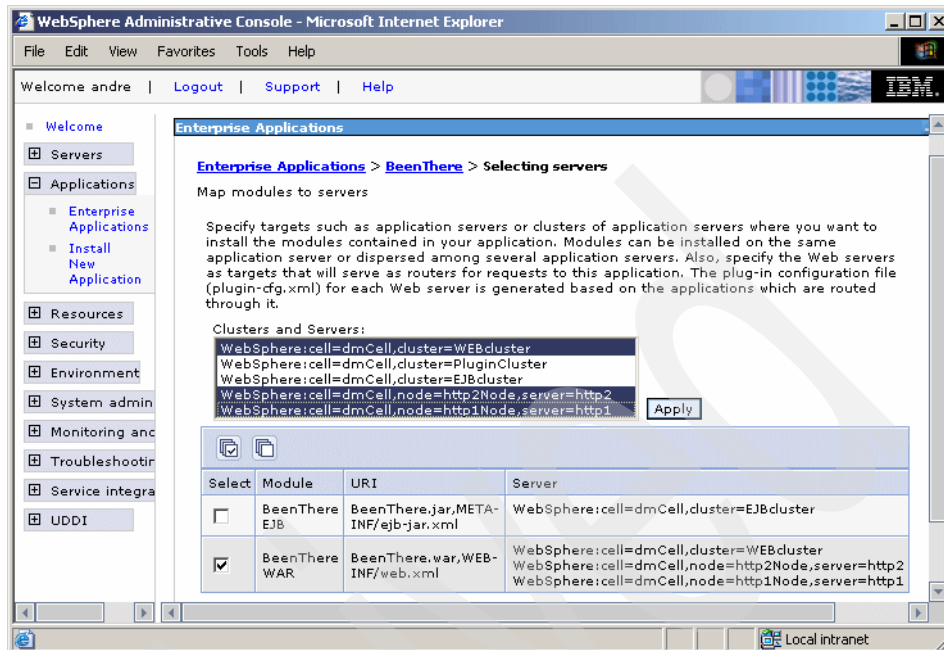


Figure 6-13 Mapping Web modules to Web servers

As a result of such a mapping, each Web server will have a specific WebSphere plug-in configuration file generated according to its individual settings and targeted applications (more on this can be found in 6.5, “WebSphere plug-in configuration file” on page 251).

6.4.1 Managed Web servers

A *managed Web server* is a Web server managed by a Node Agent running on the same node (that is, a managed node).

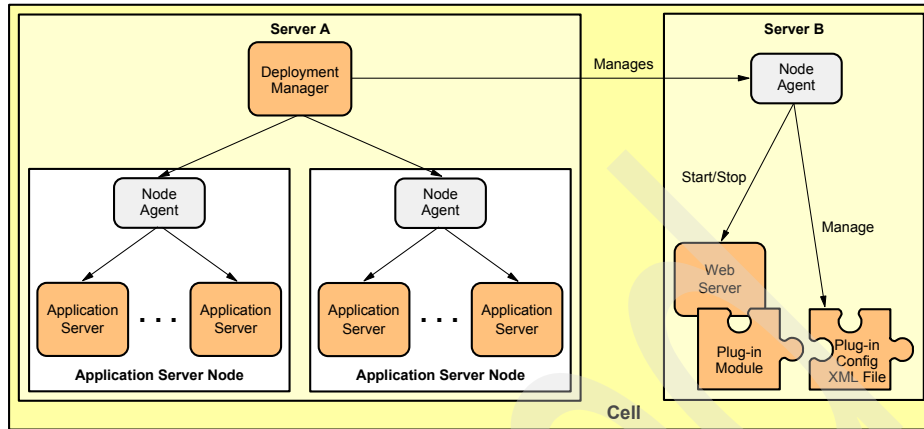


Figure 6-14 Managed Web server

In the situation shown in Figure 6-14, the Node Agent receives commands from the Deployment Manager to administer the Web server. The main administering functions are stopping and starting the Web server and the propagation of the plug-in configuration file.

Configuring a managed Web server

To create a new Web server definition for a managed node, select **Servers -> Web servers** and click **New**. This brings up the window shown in Figure 6-15 on page 248.

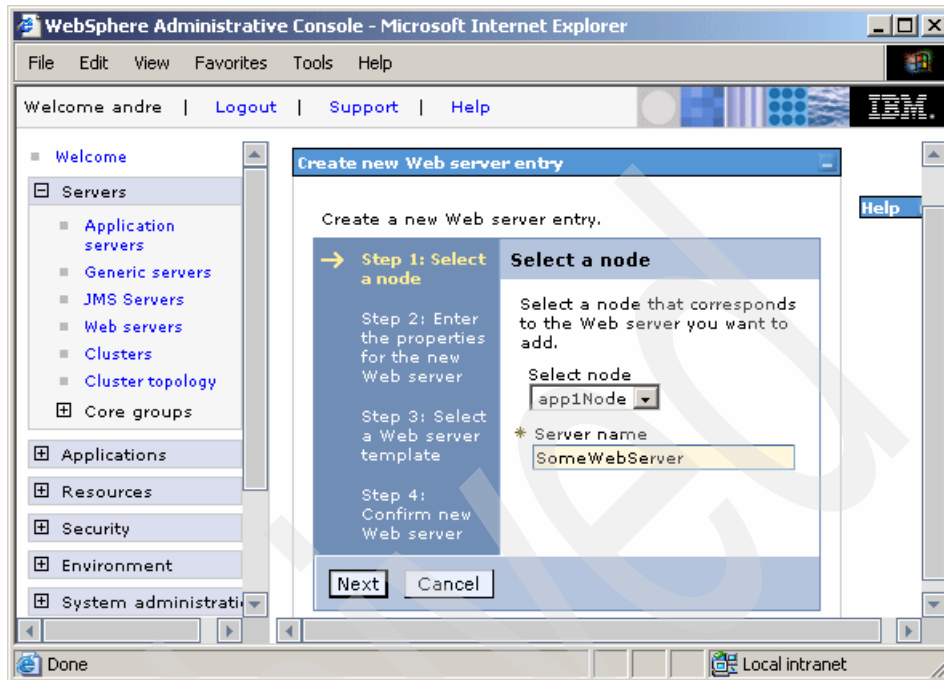


Figure 6-15 Creating a managed Web server definition

After entering the desired name and selecting an adequate node (which also hosts an application server and thus a Node Agent), click **Next**. In the next windows, you must enter the correct definitions for the already installed and configured Web server.

It is very important to keep in mind that all paths asked for are local paths of the remote managed node. Remember that, in the end, it is the remote Node Agent and the Web server plug-in that refer to these paths.

A managed Web server makes sense behind a firewall, where a WebSphere node can be installed. It is unlikely, though, in a production Web server residing in a DMZ; in this case, you will most likely be using an unmanaged node.

6.4.2 Unmanaged Web servers

Web servers running on a node without a Node Agent are called *unmanaged Web servers*. An unmanaged Web server is not managed by the Deployment Manager (just as in WebSphere Application Server V5.1; for example, it cannot be started or stopped from the Administrative Console). However, it still needs to be

configured in the cell topology and targeted by Web modules so that the Deployment Manager can generate its plug-in configuration properly.

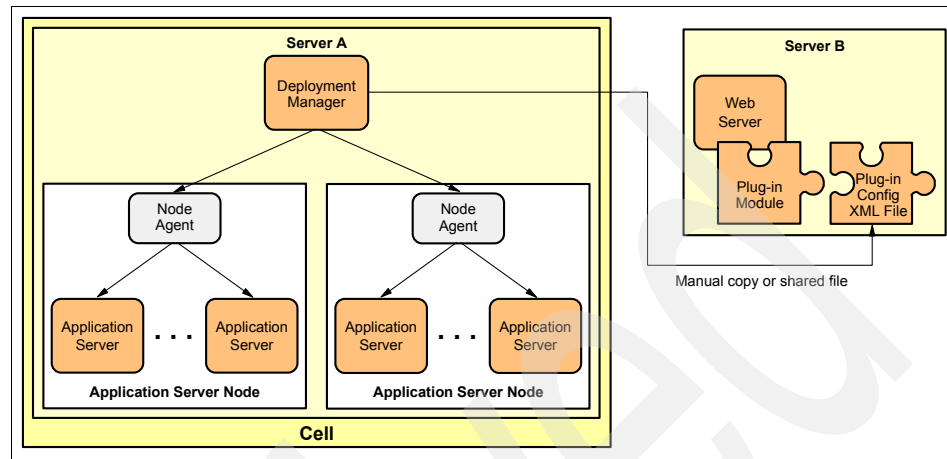


Figure 6-16 Web server on unmanaged Node

The steps for configuring an unmanaged Web server are:

1. Configuring an unmanaged node
2. Configuring an unmanaged Web server

Configuring an unmanaged node

To configure an unmanaged node, select **System administration -> Nodes** and click **Add Node**. Select the **Unmanaged node** radio button as shown in Figure 6-17 on page 250.

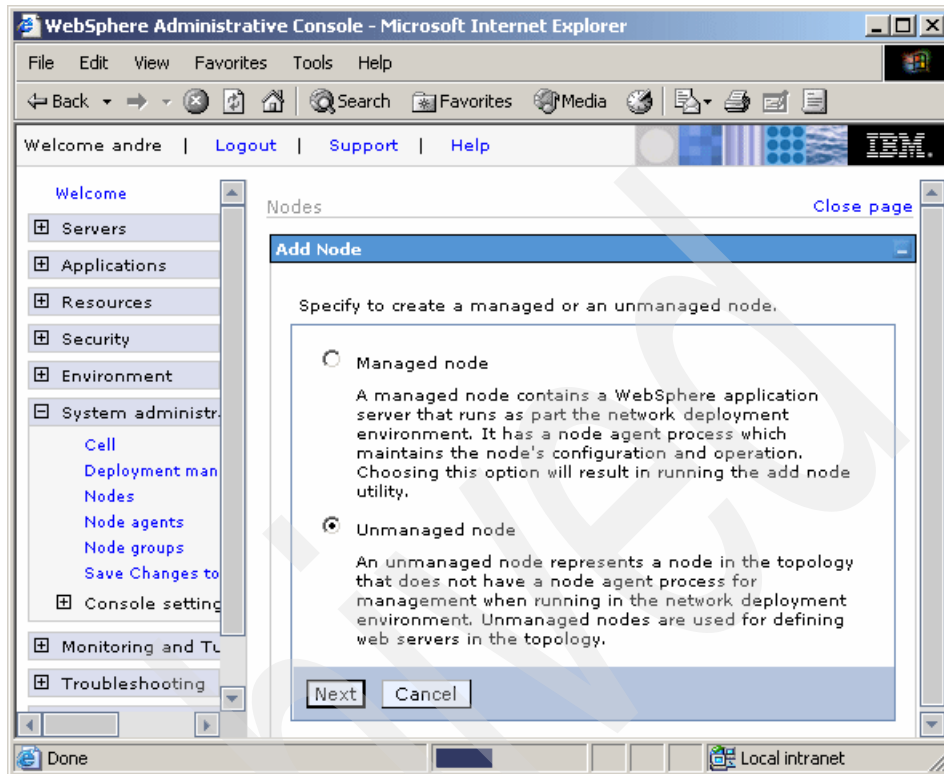


Figure 6-17 Configuring an unmanaged node

After clicking **Next**, you are asked for the name, host name and OS platform of the new node. Fill in all values and click **OK**.

Configuring an unmanaged Web server

To create a new Web server definition on the previously created node, select **Servers -> Web servers -> New**, then enter the correct definitions for the existing Web server on the previously defined node.

As mentioned earlier, remember that all paths asked for are local paths of the remote unmanaged node.

6.4.3 Unmanaged IBM HTTP Server V6.0 server (special case)

Unmanaged Web servers that run the Apache-based IBM HTTP Server V6.0 are a special type of unmanaged Web servers.

The IBM HTTP Server V6.0 Administration Server (a separate process from the Web server itself) in practice replaces the role of the Node Agent in Figure 6-14 on page 247, providing the very same services for the Deployment Manager.

Note: In fact, the IBM HTTP Server V6.0 Administration Server has been modified so that its features are only available via the WebSphere Administrative Console (in other words, you cannot connect directly to it with a Web browser).

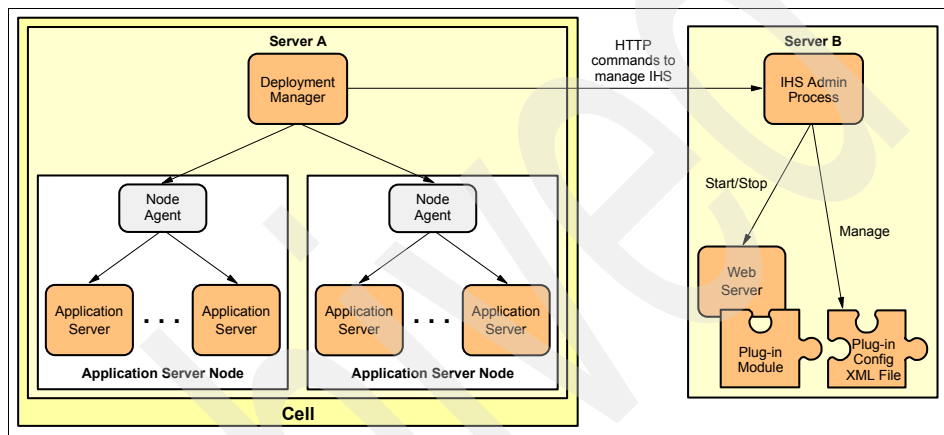


Figure 6-18 Unmanaged IBM HTTP Server - special case

The only catch is that a port must be opened on the firewall to allow the HTTP(S) communication between the Deployment Manager and the IBM HTTP Server V6.0 Administration Server HTTP port (the default is 8008).

The configuration steps needed for setting up the IBM HTTP Server V6.0 are identical to the configuration steps for an unmanaged Web server, as described in 6.4.2, “Unmanaged Web servers” on page 248.

6.5 WebSphere plug-in configuration file

In this section, we explore the plug-in configuration file: what it is composed of, how it is generated and how it is used by the WebSphere plug-in.

This section’s main topics are:

- ▶ “The plug-in configuration file” on page 252
- ▶ “Generation of the plug-in configuration file” on page 258
- ▶ “Propagation of the plug-in file” on page 262

6.5.1 The plug-in configuration file

Unlike its previous versions, WebSphere Application Server V6 is now capable of generating a specific plug-in configuration file for each Web server in the cell topology. This is possible because Web modules are now mapped not only to application servers (or clusters), but also to Web servers, as seen in Figure 6-13 on page 246.

Note: It is still possible to generate a “full” plug-in XML file like in WebSphere Application Server V5.x (a single plug-in configuration file that resolves all URIs to all applications in every Web container in the cell). If this is the desired outcome, all the administrator needs to do is to run the **GenPluginCfg.bat** (or **.sh**) script with no parameters on the Deployment Manager node. The plugin-cfg.xml generated this way must be copied manually onto each Web server, just like it used to be with WebSphere V5.x.

Naturally, a Web server whose plug-in loads such a configuration file does not need to be configured in the cell.

Example 6-2 shows a sample configuration file for a single Web server. When generating the plug-in configuration file (plugin-cfg.xml) for a Web server named http1 which is running on node http1Node, the generated file is placed in this location *on the Deployment Manager node*:

```
<WAS_HOME>/profiles/<dmProfileName>/config/cells/<cellName>/nodes/  
<nodeName>/servers/<serverName>
```

For example, in our AIX environment, this is:

```
/usr/WebSphere/AppServer/profiles/dm/config/cells/dmCell/nodes/  
http1Node/servers/http1
```

For information about how to generate a plug-in configuration file, please see 6.5.2, “Generation of the plug-in configuration file” on page 258.

The configuration file should be *propagated* (or manually copied) to its respective Web server node. We discuss the propagation of plug-in configuration files in 6.5.3, “Propagation of the plug-in file” on page 262.

The following is an example of a plug-in configuration file for a specific Web server:

Example 6-2 plugin-cfg.xml example

```
<?xml version="1.0" encoding="ISO-8859-1"?><!--HTTP server plugin config file  
for the webserver dmCell.http1Node.HTTP1 generated on 2004.10.12 at 04:43:03 PM  
EDT-->
```

```

<Config ASDisableNagle="false" AcceptAllContent="false"
AppServerPortPreference="HostHeader" ChunkedResponse="false"
IISDisableNagle="false" IISPluginPriority="High" IgnoreDNSFailures="false"
RefreshInterval="60" ResponseChunkSize="64" VHostMatchingCompat="false">
  <Log LogLevel="Error"
Name="c:\WebSphere\Plugins\logs\HTTP1\http_plugin.log"/>
  <Property Name="ESIEnable" Value="true"/>
  <Property Name="ESIMaxCacheSize" Value="1024"/>
  <Property Name="ESIInvalidationMonitor" Value="false"/>
  <VirtualHostGroup Name="default_host">
    <VirtualHost Name="*:9080"/>
    <VirtualHost Name="*:80"/>
    <VirtualHost Name="*:9081"/>
  </VirtualHostGroup>
  <ServerCluster CloneSeparatorChange="false" LoadBalance="Round Robin"
Name="WEBcluster" PostSizeLimit="-1" RemoveSpecialHeaders="true"
RetryInterval="60">
    <Server CloneID="vtnu14vu" ConnectTimeout="0" ExtendedHandshake="false"
LoadBalanceWeight="2" MaxConnections="-1" Name="app1Node_Web1"
WaitForContinue="false">
      <Transport Hostname="app1.itso.ibm.com" Port="9080" Protocol="http"/>
    </Server>
    <Server CloneID="vtnu19n9" ConnectTimeout="0" ExtendedHandshake="false"
LoadBalanceWeight="2" MaxConnections="-1" Name="app2Node_Web2a"
WaitForContinue="false">
      <Transport Hostname="app2.itso.ibm.com" Port="9080" Protocol="http"/>
    </Server>
    <Server CloneID="vtnu1d27" ConnectTimeout="0" ExtendedHandshake="false"
LoadBalanceWeight="3" MaxConnections="-1" Name="app2Node_Web2b"
WaitForContinue="false">
      <Transport Hostname="app2.itso.ibm.com" Port="9081" Protocol="http"/>
    </Server>
  <PrimaryServers>
    <Server Name="app1Node_Web1"/>
    <Server Name="app2Node_Web2a"/>
    <Server Name="app2Node_Web2b"/>
  </PrimaryServers>
</ServerCluster>
  <UriGroup Name="default_host_WEBcluster_URIs">
    <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid"
Name="/trade/*"/>
  </UriGroup>
  <Route ServerCluster="WEBcluster" UriGroup="default_host_WEBcluster_URIs"
VirtualHostGroup="default_host"/>
  <RequestMetrics armEnabled="false" loggingEnabled="false" rmEnabled="false"
traceLevel="HOPS">
    <filters enable="false" type="URI">
      <filterValues enable="false" value="/snoop"/>
      <filterValues enable="false" value="/hitcount"/>
    </filters>
  </RequestMetrics>
</Config>

```

```

</filters>
<filters enable="false" type="SOURCE_IP">
  <filterValues enable="false" value="255.255.255.255"/>
  <filterValues enable="false" value="254.254.254.254"/>
</filters>
<filters enable="false" type="JMS">
  <filterValues enable="false" value="destination=aaa:topic=bbb"/>
</filters>
<filters enable="false" type="WEB_SERVICES">
  <filterValues enable="false"
value="wsdlPort=aaa:op=bbb:nameSpace=ccc"/>
</filters>
</RequestMetrics>
</Config>

```

The main tags within this file are listed in Table 6-2, and can be associated with the flow chart shown in Figure 6-23 on page 265 to help understand what each is defining.

Table 6-2 Plug-in configuration XML tag descriptions

No.	XML Tag	Description
2,3	VirtualHostGroup VirtualHost	A group of virtual host names and ports that will be specified in the HTTP Host header when the user tries to retrieve a page. Enables you to group virtual host definitions together that are configured to handle similar types of requests. The requested host and port number are matched to a VirtualHost tag in a VirtualHostGroup.
2,4	UriGroup Uri	A group of URIs that will be specified on the HTTP request line. The incoming client URI is compared with all the Uri tags in the UriGroup to see if there is a match to determine if the application server will handle the request for the Route in conjunction with a virtual host match.

No.	XML Tag	Description
2,3,5,6	Route	<p>The Route definition is the central element of the plug-in configuration. It specifies how the plug-in will handle requests based on certain characteristics of the request. The Route definition contains the other main elements: a required ServerCluster, and either a VirtualHostGroup, UriGroup, or both.</p> <p>Using the information that is defined in the VirtualHostGroup and the UriGroup for the Route, the plug-in determines if the incoming request to the Web server should be sent on to the ServerCluster defined in this Route.</p> <p>The plug-in sets scores for Routes if there is a VirtualHost and Uri match for an incoming request. Once the plug-in processes all Routes, the Route chosen is the one with the highest score.</p>
6	ServerCluster Server	The located ServerCluster from the Route tag contains a list of Server tags that in turn contain the requested object. At 7 , the Server tag is used to check session affinity. At 8 or 9 , the correct server is selected.
8,9	ServerCluster Server	The ServerCluster located by finding the correct Route can optionally specify the WLM algorithm. This will then be used to select one Server from within the ServerGroup.
10	Transport	Once a Server has been located, its Transport tags describe how to connect to it.

Tip: A detailed description of each XML tag and its relationships can be found in the WebSphere V6 InfoCenter at:

<http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp>

To find the appropriate section, search for “plugin-cfg.xml file”.

Plug-in file tags related to workload management

There are some settings in the plug-in file that directly affect how the plug-in works in a workload management environment. In WebSphere V6, all of these settings can be modified using the Administrative Console:

- You can change the workload distribution policy in the configuration file. See 6.6.2, “Plug-in workload management policies” on page 267.

- ▶ You can change the retry interval for connecting to a cluster member marked as down.
- ▶ You can change the refresh interval for the reloading of the plug-in configuration file. The refresh interval defines how often the plug-in will check to see if the configuration file has changed. See “Refresh interval of the plug-in” on page 263.
- ▶ You can change the maximum number of connections that will be allowed to a server from a given plug-in. If this attribute is set to either zero or -1, there is no limit to the number of pending connections to the application servers. The default value is -1.

Changing plugin-cfg.xml settings

It is important to understand that manually editing the plug-in configuration files is not necessary (nor recommended) anymore. Almost all settings can now be changed through the Administrative Console (in various configuration panels). For example, all application server related settings can be found in **Servers -> Application servers -> <AppServer_Name> -> Web Server plug-in properties**, as shown in Figure 6-19 on page 257.

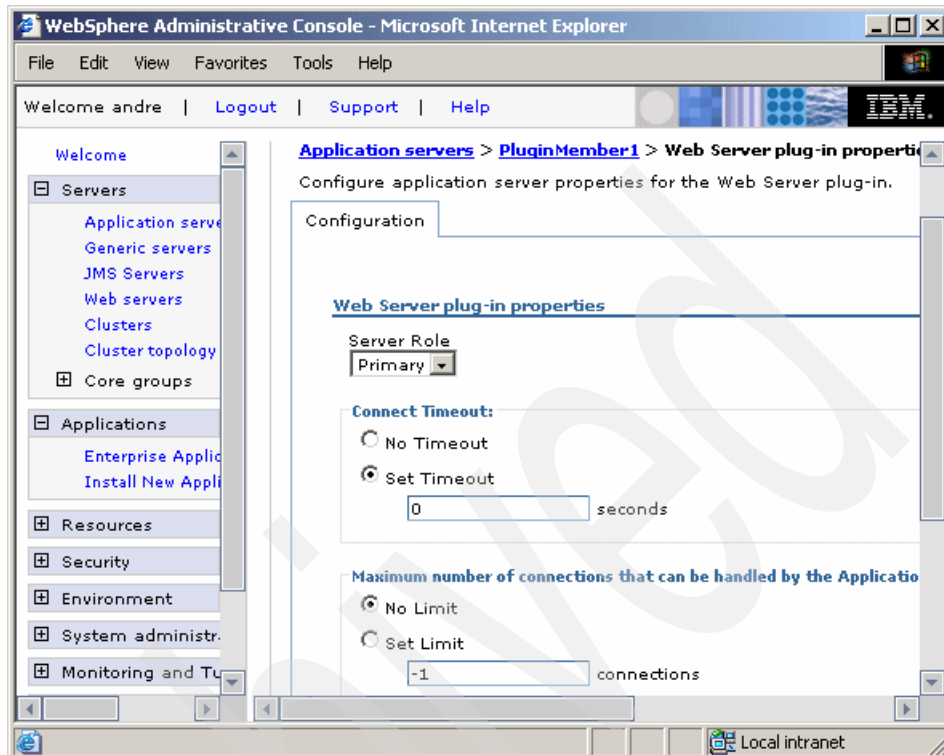


Figure 6-19 Plug-in properties (application server level)

The Web server related settings can be found in **Servers -> Web Servers -> <WebServer_Name> -> Plug-in properties**, as detailed in Figure 6-20 on page 258.

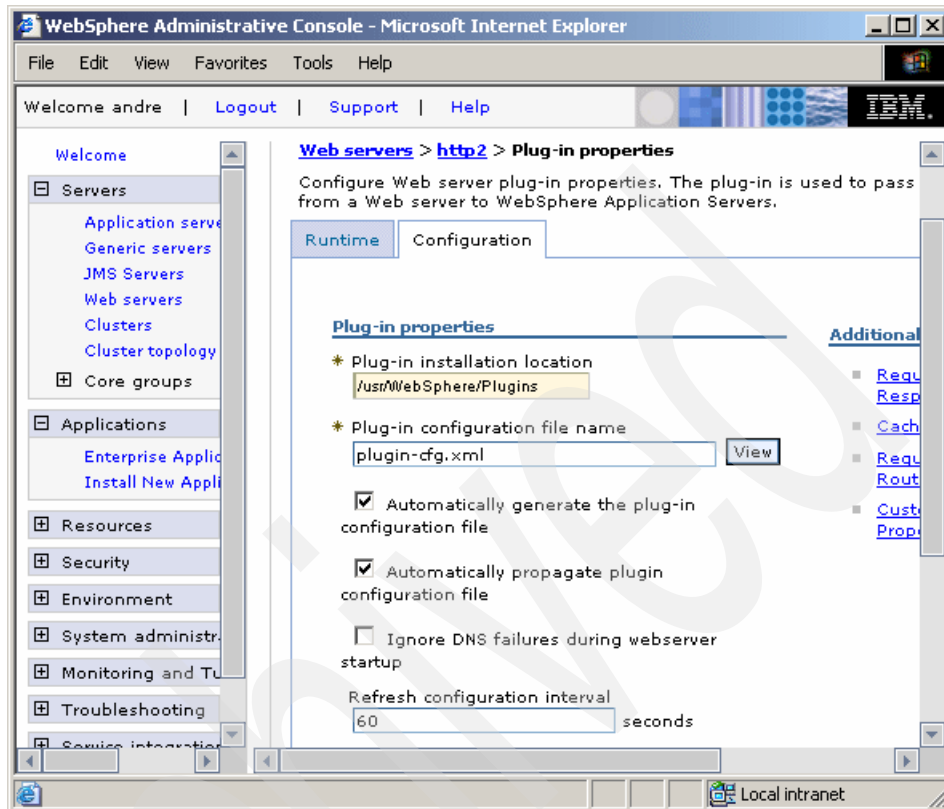


Figure 6-20 Plug-in properties for Web servers

The only exception that we found during our testing is the ClusterAddress tag which cannot be set using the Administrative Console. Please refer to “ClusterAddress” on page 271 for information about this setting.

6.5.2 Generation of the plug-in configuration file

Generating the plug-in configuration for a Web server recreates its respective plugin-cfg.xml using the current settings of objects in the cell. Once the plug-in has reloaded its new configuration, clients are able to access the updated or new Web resources.

WebSphere Application Server V6 can automatically regenerate (and propagate) the plugin-cfg.xml file each time a related setting is changed. This is the default option, as can be seen on Figure 6-20, for the *Automatically generate...* checkbox. For this setting to work, however, the *Web server plug-in configuration*

service must be enabled (see Figure 6-21), by selecting **System Administration -> Deployment manager -> Administration Services -> Web server plug-in configuration service** and checking the **Enable automated Web server configuration processing** checkbox.

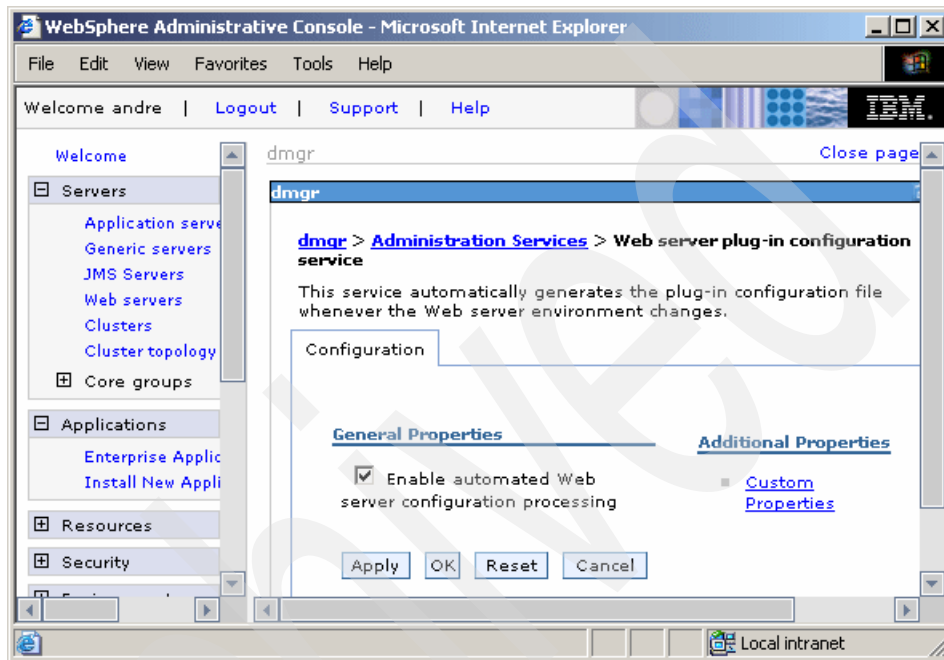


Figure 6-21 Web server plug-in configuration service

So the Web server's default configuration (together with the setting just mentioned) ensures that its plug-in configuration file will be regenerated automatically whenever necessary (for example, when mapping a new Web module to it). On the other hand, you can always regenerate the file manually when you wish to do so.

As mentioned earlier, a generated plug-in configuration file is stored at the following location:

```
<WAS_HOME>/profiles/<dmProfileName>/config/cells/<cellName>/nodes/  
<nodeName>/servers/<serverName>
```

Manual regeneration of a plug-in file

Manual regeneration of a plug-in file can be performed at any time, whether application servers are running or not. The result of its propagation (or manual copy) and delay are discussed in 6.5.3, “Propagation of the plug-in file” on page 262. There are two methods of manual regeneration:

- ▶ From the Administrative Console
- ▶ Via a command line using the **GenPluginCfg** command

Note: As mentioned earlier, it is still possible to generate a “full” plug-in configuration file so that a plug-in can redirect requests to all Web containers in the cell (just like in WebSphere Application Server V5.x) by running the **GenPluginCfg** script with no parameters on the Deployment Manager node.

Executing the same script (also without parameters) on any running node other than the Deployment Manager results in a `plugin-cfg.xml` file that contains a subset of the cell-wide configuration (serving all Web modules on the node). This is the result of each node only having a subset of the master configuration repository replicated locally.

From the Administrative Console

The Administrative Console can be used to regenerate the plug-in for one or more Web servers. In a Network Deployment environment, the plug-in is regenerated on the Deployment Manager system and, depending on the overall configuration, propagated to the Web servers. This means that every Web server configuration in the cell is relevant (even for unmanaged non-IBM HTTP Server servers), as there are directory paths in the `plugin-cfg.xml` file that must correspond to real paths on the machine this file is copied or propagated to.

To regenerate the plug-in file manually from the Administrative Console:

1. Select **Servers -> Web Servers**.
2. Select the appropriate Web server(s) as shown in Figure 6-22 on page 261.
3. Click **Generate Plug-in**. A message appears to inform you whether the plug-in regeneration for each Web server was successful.

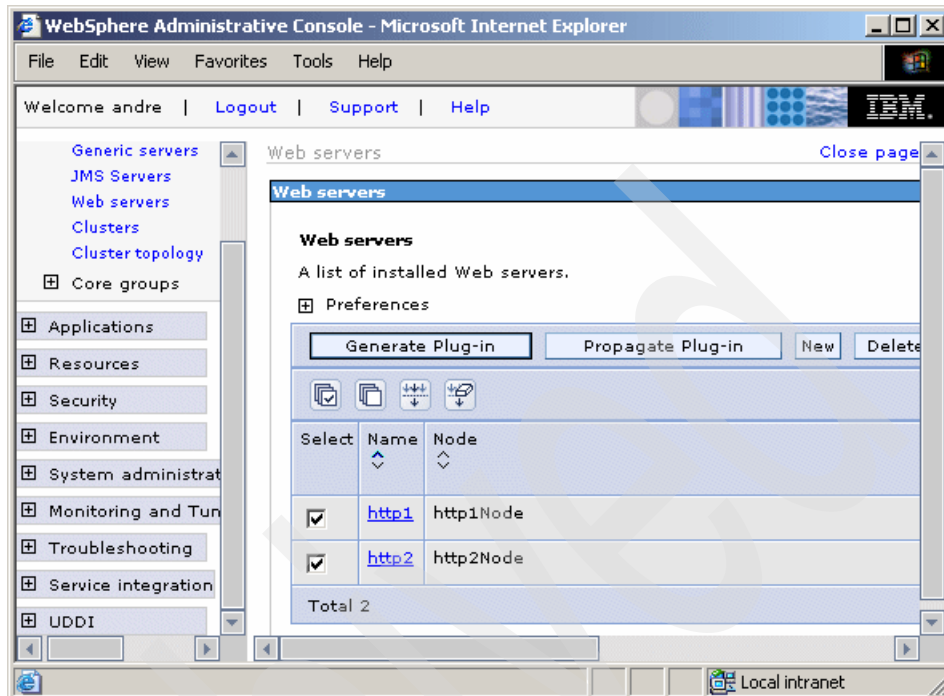


Figure 6-22 Regenerating the plug-in using the Administrative Console

Using the **GenPluginCfg** command

The command **GenPluginCfg** can be used in its simplest form by typing:

`GenPluginCfg.sh` (for UNIX environments)

or

`GenPluginCfg.bat` (for Windows environments)

This generates the plug-in configuration file on the node where you are running the command (the resulting file will consider all deployed Web modules and configured Web containers on this node). However, it is recommended that you run this command on the Deployment Manager node, since the resulting file then considers the entire cell configuration. In other words, when generating the plug-in configuration via **GenPluginCfg** on a node other than the Deployment Manager node, the resulting file only contains the local node configuration.

The command **GenPluginCfg** can also be used for fine-grained work, generating a plug-in configuration file for a single Web server by adding the appropriate parameters. For example:

```
GenPluginCfg.sh -cell.name dmCell -node.name http1Node  
-webserver.name http1
```

There are many other command-line tools available in WebSphere Application Server V6. For details on the **GenPluginCfg** command as well as other available commands, refer to Chapter 6, “Administration with scripting”, of the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

6.5.3 Propagation of the plug-in file

A new feature in WebSphere Application Server V6 is the propagation of a plug-in configuration file from the Deployment Manager node to its respective Web server node. This eliminates the need to manually copy these files to the Web server; since Web servers usually reside in a segregated network (the DMZ), the simple task of copying files can be sometimes tricky and cumbersome.

The propagation of a plug-in configuration file is only possible for managed Web servers and for unmanaged IBM HTTP Server V6.0 servers. For other unmanaged Web servers, the plug-in configuration file must still be copied manually to the Web server system.

Depending on the current Web server configuration (see Figure 6-20 on page 258, especially the *Automatically propagate...* checkbox) a recently regenerated plug-in configuration file can also be propagated automatically to its respective Web server node.

To manually propagate a plug-in configuration file:

1. Select **Servers -> Web Servers**.
2. Select the appropriate Web server(s), as shown in Figure 6-22 on page 261.
3. Click **Propagate Plug-in**. A message appears to inform you whether the plug-in propagation for each Web server was successful.

You must specify the plug-in installation location during configuration of a Web server in the cell (for example, c:\websphere\plugins). When propagating the plug-in configuration file, it is then placed into the config directory found under this previously specified directory on the Web server's file system, normally:

```
<PLUGIN_HOME>/config/<WebServerName>
```

For our Web server on Windows, this is:

```
C:\WebSphere\Plugins\config\http1
```

If you copy the configuration file(s) manually, they must be copied to that very same location.

After propagation (or manual copy) of a plug-in configuration file there will be a short delay before in the changes take effect. This delay is governed by the refresh interval of the plug-in.

Refresh interval of the plug-in

The WebSphere plug-in checks for a new configuration file only at specified time intervals. This interval (in seconds) is determined by the Refresh configuration interval setting seen in Figure 6-20 on page 258 (which represents the RefreshInterval attribute of the Config tag in the plugin-cfg.xml file shown in Example 6-3). If a newly generated configuration file has been propagated (or manually copied) to a Web server node, then there can be a delay of anything up to the number specified in this setting before the plug-in will actually load and use the new configuration.

Example 6-3 Plugin-cfg.xml extract showing RefreshInterval

```
<?xml version="1.0" encoding="ISO-8859-1"?><!--HTTP server plugin config file
for the webserver dmCell1.http2Node.http2 generated on 2004.10.12 at 04:43:03 PM
EDT-->
<Config ASDisableNagle="false" AcceptAllContent="false"
AppServerPortPreference="HostHeader" ChunkedResponse="false"
IISDisableNagle="false" IISPluginPriority="High" IgnoreDNSFailures="false"
RefreshInterval="60" ResponseChunkSize="64" VHostMatchingCompat="false">
...
```

Once the RefreshInterval has passed, the actual loading of the new plug-in configuration file into the plug-in runtime is triggered by a request to the Web server.

In a development environment in which changes are frequent, a lower setting than the default setting of 60 seconds might be useful. Once in production, changing this to a higher value than the default is recommended because updates to the configuration normally do not occur as often.

Note: If a change is made to the virtual host(s) that a Web module uses, a generated plug-in file will be aware of that. However, you should also restart the related application servers so that these also reflect the change.

6.5.4 Bypassing the plug-in

If you are working in a development environment, where constant changes are being made to installed applications, you could use the WebContainer Inbound Chain of a specific Application Server to access your application directly instead of going through a Web server. For example, pointing your browser to `http://myhost:9080/snoop` will bypass the plug-in.

If that is intended, the HTTP ports of the Web container transport chains must be added to the proper virtual hosts aliases.

When things go wrong: Even in a production environment, it is sometimes useful to bypass the plug-in for testing purposes.

For example, in our sample topology, a request from a client browser goes through a Caching Proxy, a Load Balancer, a Web server (and its plug-in) and finally arrives at the Web container. When things stop working, it is always recommended that you send a request directly to the Web container to find out quickly if it is still responding.

6.6 WebSphere plug-in workload management

The actual management of Web container workload is performed by the Web server plug-in. This section describes how the plug-in routes requests to the correct Web containers and how the plug-in is configured. Understanding how the plug-in makes these decisions is key to understanding how WebSphere workload manages Web requests.

6.6.1 Processing requests

Once a cluster and its members have been set up, the plug-in can start directing requests from the Web server to the correct application server. Figure 6-23 on page 265 shows what occurs within the plug-in when processing a request.

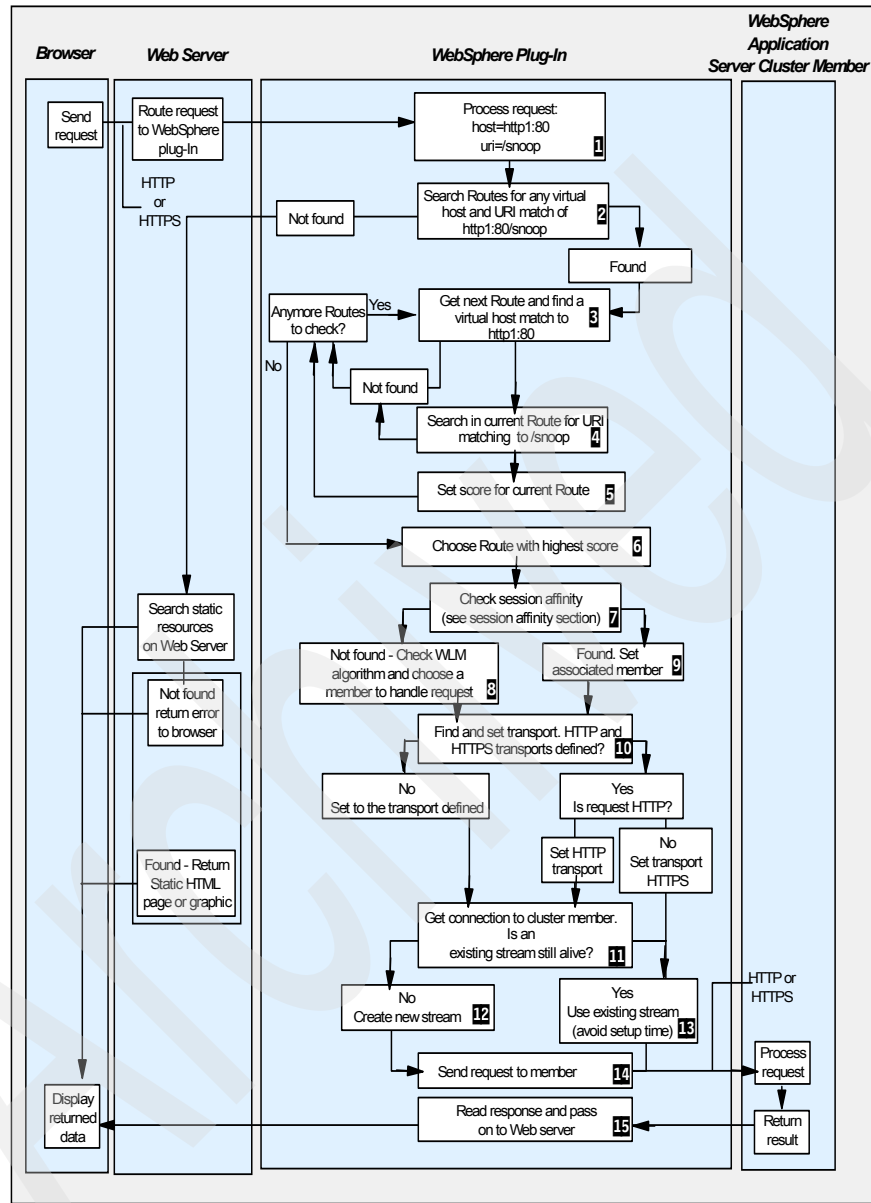


Figure 6-23 How the plug-in processes a request

Here is an example walk-through of Figure 6-23 on page 265. Users asks for the page `http://http1:80/snoop` from their browser. The request is routed to the Web server over the Internet.

1. The Web server immediately passes the request to the plug-in (1). All requests go to the plug-in first.
2. The plug-in then starts by looking at all Route definitions in the `plugin-cfg.xml`. For each Route, it searches through its configuration to find if there is any match to the virtual host `http1:80` and URI `/snoop`. It will find the first match and then decide that WebSphere should handle the request (2). If there is no match, WebSphere will not handle the request.
3. The plug-in takes the request and separates the host name and port pair and URI. The plug-in now starts by looking at all the defined Routes. It gets a Route and then searches for a virtual host match to `http1` port `80` in that Route. It matches that host name and port to `http1:80` in the VirtualHost block (3).
4. The plug-in then tries to match the URI `/snoop` in the current Route. It searches its configuration for a URI mapping that matches the requested URI in the UriGroup block (4). It matches `/snoop` in the UriGroup.
5. Once the VirtualHostGroup and UriGroup are found, it sets a score depending on the number of characters in the URI and virtual host (5).
6. The plug-in continues to the next Route and searches through virtual hosts and URI's setting scores for matches with any URI and virtual host match. The Route that has the highest score is chosen (6) and the ServerCluster is set.
7. The plug-in now checks the request to see if any session identification has been passed to the server (7). See 6.8, "HTTP session management" on page 279 for more information about this. Our request does not contain any session information.
8. The plug-in chooses a cluster member to manage the request (8). See 6.6.2, "Plug-in workload management policies" on page 267 for more information. A server is chosen to handle the request. If there is a session identifier and a CloneID associated with it, the plug-in will choose a server based on the CloneID (9).
9. This cluster member has two transports associated with it: HTTP and HTTPS are defined. Because this request is HTTP, the cluster member uses the HTTP transport definition (10).
10. In box 11 the term *stream* is used. A stream is a persistent connection to the Web container. Since HTTP 1.1 is used for persistent connections between the plug-in and Web container, it is possible to maintain a connection (stream) over a number of requests. In our example, no previous connection has been created from the plug-in to the Web container, so a new stream is created

(12). However, if a stream is already established, the plug-in uses the existing stream (13).

11. The request is sent to the cluster member (14) and successfully processed. The plug-in passes the response on to the Web server (15), which in turn passes it back to the user's browser.

6.6.2 Plug-in workload management policies

This section describes the process that the plug-in goes through to choose a cluster member to serve a request. We cover the algorithms used to choose a cluster member and the various ways this process can be manipulated. We also take a deeper look at how the plug-in connects to each application server. With this knowledge, it is possible to understand how failover situations are dealt with.

Workload management policies

Since WebSphere Application Server V5, the plug-in has two options for the load distributing algorithm:

- ▶ Round robin with weighting
- ▶ Random

Note: The weighting for the round robin approach can be turned off by giving all application servers in a cluster equal weights.

The default value is Round Robin. It can be changed by selecting **Servers -> Web Servers -> <WebServer_Name> -> Plug-in properties -> Request Routing** as shown in Figure 6-24 on page 268.

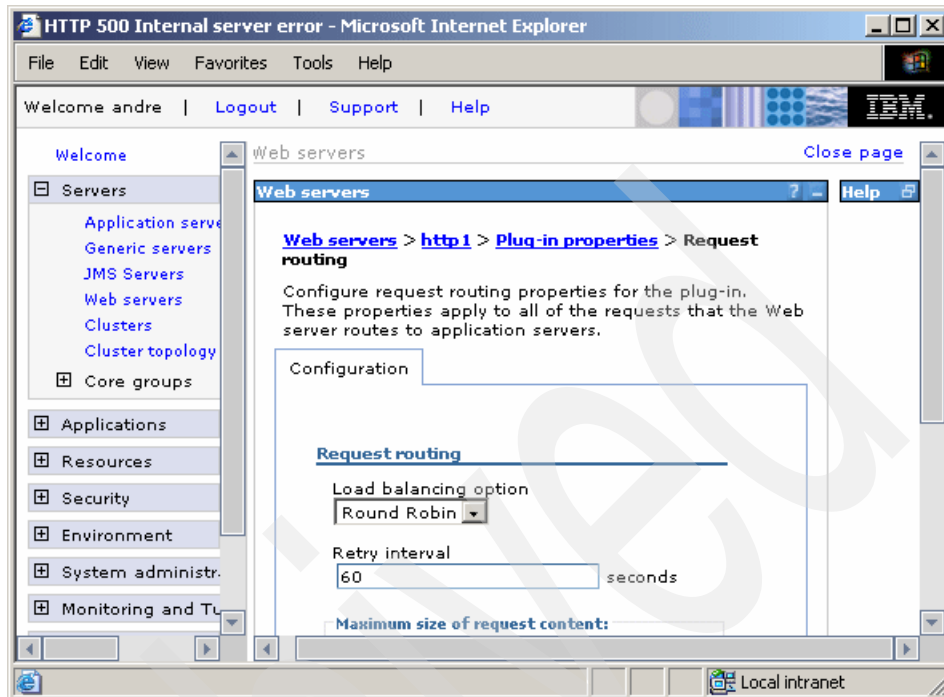


Figure 6-24 Plug-in load balancing options

There is also a feature in WebSphere Application Server V6 for the plug-in called `ClusterAddress` that can be used to suppress load balancing. See “`ClusterAddress`” on page 271 for more details.

For examples of these workload management policies, please refer to 6.10.1, “Normal operation” on page 310.

Weighted round robin

When using this algorithm, the plug-in selects a cluster member at random from which to start. The first successful browser request is routed to this cluster member and then its weight is decremented by 1. New browser requests are then sent round robin to the other application servers and subsequently the weight for each application server is decremented by 1. The spreading of the load is equal between application servers until one application server reaches a weight of 0. From then on, only application servers with a weight higher than 0 will have requests routed to them. The only exception to this pattern is when a cluster member is added or restarted or when session affinity comes into play.

Notes:

- ▶ We are only using two application servers (Web1 and Web2a) in this example to make it easier to explain.
- ▶ The first request goes to a random application server to avoid multiple Web servers and/or multi-process Web servers directing initial requests to the same application server.
- ▶ When starting the HTTP Server, the application server weight is reduced to the lowest common denominator. For example: Web1's weight is 8 and Web2a's weight is 6. When you start the HTTP Server, the weight of Web1 is set to 4 and the weight of Web2a is set to 3. This is done to avoid application servers with a lower weight to be idle for long.

Using Table 6-3 on page 270 and the following explanations, we show you how weighted round robin is performed. To begin with, we have a weight of 4 for Web1 and a weight of 3 for Web2a:

1. When the first request comes in, Web1 is randomly chosen. The request is OK. Web1 weight is decremented by 1 to 3.
2. The second request is sent to Web2a. The request is OK. Web2a weight is decremented by 1 to 2.
3. The third and fourth requests are sent to Web1 and Web2a, respectively. So Web1 now has a weight of 2 and Web2a now has a weight of 1.
4. The fifth request has a cookie that specifies a server affinity to Web1. The request is sent to Web1 and its weight is decremented by 1 to 1.
5. The sixth request is again sent to Web1 because of server affinity. The request is OK. Web1's weight is decremented by 1 to 0.
6. The seventh request again has a cookie that specifies server affinity to Web1. The request is sent to Web1 and its weight is decremented by 1 to -1.
7. The eighth to eleventh request all have server affinity to Web1. The weight is decremented by 1 for each request. After the eleventh request, Web1 now has a weight of -5 while Web2a still has a weight of 1.
8. The twelfth request has no server affinity so it is sent to Web2a. Web2a weight is decremented by 1 to 0.
9. When processing the thirteenth request, the plug-in decides to reset the weights because there are no servers marked up having positive weights. A multiple of the lowest common denominator of the servers' maximum weight is added back to the current weights to make all weights positive again. See the Important shaded box below for a detailed description of how the weights are reset.

After resetting the weights, the sequence is repeated with the same starting point (there is no random server selection this time); in our case, this means that the thirteenth request is sent to Web1 (after the weights have been reset) and Web1's weight is decremented by 1 to 2.

Important: In our example, the current weight of Web1 is -5 because many session-based requests have been served. Web2a has a weight of 0. The plug-in checks how many times the maxWeight should be added to make the current weight positive for all servers. The starting weight for Web1 was 4 and 3 for Web2a. Because Web1's current weight is -5, adding 4 (the lowest common denominator) would not set it to a positive weight. Thus the plug-in decides to add the starting weights * 2, which is 8 for Web1 and 6 for Web2a. So the new current weights are 3 for Web1 ($-5 + 2 * 4$) and 6 for Web2a ($0 + 2 * 3$).

Table 6-3 Request handling using weighted round robin server selection

Number of Requests	Web1 Weight	Web2a Weight
0	4	3
1	3	3
2	3	2
3	2	2
4	2	1
5	1	1
6 - Request with session affinity to Web1	0	1
7 - Request with session affinity to Web1	-1	1
8 - Request with session affinity to Web1	-2	1
9 - Request with session affinity to Web1	-3	1
10 - Request with session affinity to Web1	-4	1
11 - Request with session affinity to Web1	-5	1

Number of Requests	Web1 Weight	Web2a Weight
12 - No session affinity for this request	-5	0
13 RESET - no session affinity for this request	2	6

Random

Requests are passed to cluster members randomly. Weights are not taken into account as with round robin. The only time the application servers are not chosen randomly is when there are requests with sessions associated with them. When the random setting is used, cluster member selection does not take into account where the last request was handled. This means that a new request could be handled by the same cluster member as the last request.

Suppressing workload management

There are two options to suppress plug-in workload management:

- ▶ By setting all servers to equal weights
- ▶ By adding the ClusterAddress tag to the plugin-cfg.xml file

ClusterAddress

As mentioned earlier, the plugin-cfg.xml tag called ClusterAddress can be used to suppress plug-in based load balancing. You define a ClusterAddress when you do *not* want the plug-in to perform any load balancing because you already have some type of load balancer in between the plug-in and the application servers, which can be a software or a hardware solution. The ClusterAddress specified is the IP address of your external load balancing solution. When doing so, the plug-in will only focus on route determination and session affinity. Once ClusterAddress is defined, if a request comes in that does not have session affinity established, the plug-in routes it to the ClusterAddress. If affinity has been established, then the plug-in routes the request directly to the appropriate application server, bypassing the ClusterAddress entirely. If no ClusterAddress is defined for the ServerCluster, then the plug-in load balances across the PrimaryServers list.

Note: Suppressing the plug-in load balancing is normally not desirable. As in the sample topology, a “low-level” load balancer (software or hardware-based) normally precedes the Web servers and not the application servers.

Unfortunately, the ClusterAddress configuration cannot be done using the WebSphere Administrative Console. This is a special case where you have to edit the plug-in configuration file directly.

Important: When editing plug-in configuration files directly, do so on the Deployment Manager machine. Please remember that each Web server has a specific location for its plug-in file.

The ClusterAddress tag has the same attributes and elements as a Server element. The difference is that you can only define one of them within a ServerCluster, and you should also remove all subsequent Server tags.

An example of how to configure the ClusterAddress tag in the plugin-cfg.xml is shown in Example 6-4. This example also shows the parts that should be deleted (striked through).

Example 6-4 Extract of plugin-cfg.xml showing ClusterAddress

```
.  
<ServerCluster CloneSeparatorChange="false" LoadBalance="Round Robin"  
Name="WEBcluster" PostSizeLimit="-1" RemoveSpecialHeaders="true"  
RetryInterval="120">  
  <ClusterAddress Name="MyOwnLoadBalancer">  
    <Transport Hostname="myownlb.itso.ibm.com" Port="80" Protocol="http"/>  
  </ClusterAddress>  
  <Server CloneID="vve2m4fh" ConnectTimeout="5" ExtendedHandshake="false"  
LoadBalanceWeight="2" MaxConnections="1" Name="app1Node_Web1"  
WaitForContinue="false">  
    <Transport Hostname="app1.itso.ibm.com" Port="9080" Protocol="http"/>  
  </Server>  
</ServerCluster>  
.
```

6.7 Web container failures and failover

As mentioned previously, a complete WebSphere environment may include several Web server instances as well as several WebSphere Application Server instances. A WebSphere administrator can create any number of application server instances as cluster members. These cluster members can all reside on a single node or can be distributed across multiple nodes in the WebSphere cell (vertical or horizontal scaling).

Cluster members share application workload and provide failover support between them. You may have thin Web clients or/and thick Java/C++ clients. When using clustered WebSphere Application Servers, your clients can be redirected either automatically or manually (depending on the nature of the failure) to another healthy server in the case of a failure of a clustered application server.

In this chapter, we address only the thin Web client workload management and failover which is fully implemented by the WebSphere plug-in running within each Web server.

When a HTTP request reaches the Web server, it usually is passed to a Web container running in a WebSphere Application Server (although some administrators might want to set up an environment where static content is handled directly by the Web servers or by some caching solution). The WebSphere Web server plug-in, which runs in-process with the Web server itself, is responsible for deciding which Web container the request should be passed to. For these WebSphere requests, high availability of the Web container becomes an important piece of the environment.

The Web container failover support in WebSphere Application Server V5 and higher is provided by three mechanisms:

- ▶ A Web container server cluster, which creates server process redundancy for failover support.
- ▶ The workload management routing technique built into the Web server plug-in. It controls the routing of client requests among redundant server processes.
- ▶ Session management and failover mechanism, which provides HTTP session data for redundant server processes.

As we can see, satisfactory failover support for Web clients can only be achieved by the use of all these three mechanisms.

There are a number of situations when the plug-in might not be able to complete a request to a specific application server. In a clustered environment with several cluster members, this does not mean an interruption of the service.

Here are some example scenarios when the plug-in cannot connect to a cluster member:

1. Expected application server failures (the cluster member has been brought down intentionally for maintenance, for example).
2. Unexpected server process failures (the application server JVM has crashed, for example).
3. Server network problems between the plug-in and the cluster member (a router is broken, for example).
4. System problems (expected or not), like system shutdown or power failures.
5. The cluster member is overloaded and cannot process the request.

When the plug-in has selected a cluster member to handle a request (see Figure 6-23 on page 265, boxes 7, 8 and 9), it will attempt to communicate with the cluster member. If this communication is unsuccessful or breaks, then the plug-in will mark the cluster member as down and attempt to find another cluster member to handle the request.

In the first two failure cases described above, the physical machine where the Web container is supposed to be running is still available, although the WebContainer Inbound Chain is not available. When the plug-in attempts to connect to the WebContainer Inbound Chain to process a request for a Web resource, the machine will refuse the connection, causing the plug-in to mark the application server as down.

In the third and fourth events, however, the physical machine is no longer available to provide any kind of response. In these events, if non-blocking connection is not enabled, the plug-in waits for the local operating system to time out the request before marking the application server unavailable. While the plug-in is waiting for this connection to time out, requests routed to the failed application server appear to hang. The default value for the TCP timeout varies based on the operating system. While these values can be modified at the operating system level, adjustments should be made with great care. Modifications may result in unintended consequences in both WebSphere and other network dependent applications running on the machine. This problem can be eliminated by enabling non-blocking connection. Refer to “Connection Timeout setting” on page 335 for more information.

In the fifth case, client overloading can make a healthy server unavailable and cause a server overloading failover. This is explained in “Overloading a cluster member” on page 334.

The marking of the cluster member as down means that, should that cluster member be chosen as part of a workload management policy or in session affinity, the plug-in will not try to connect to it. The plug-in knows that it is marked as down and ignores it.

The plug-in waits for a period of time before removing the marked as down status from the cluster member. This period of time is called the *retry interval*. By default, the retry interval is 60 seconds. If you turn on tracing in the plug-in log file, it is possible to see how long is left until the cluster member will be tried again. To set the retry interval, enter the proper value for the Web server plug-in as was shown in Figure 6-24 on page 268. Refer to “Retry interval setting” on page 337 for more information.

By marking the cluster member as down, the plug-in does not spend time at every request attempting to connect to it again. It will continue using other available cluster members without retrying the downed cluster member, until the retry interval has elapsed.

Figure 6-25 shows how this selection process works. It is an expansion of box 8 from Figure 6-23 on page 265.

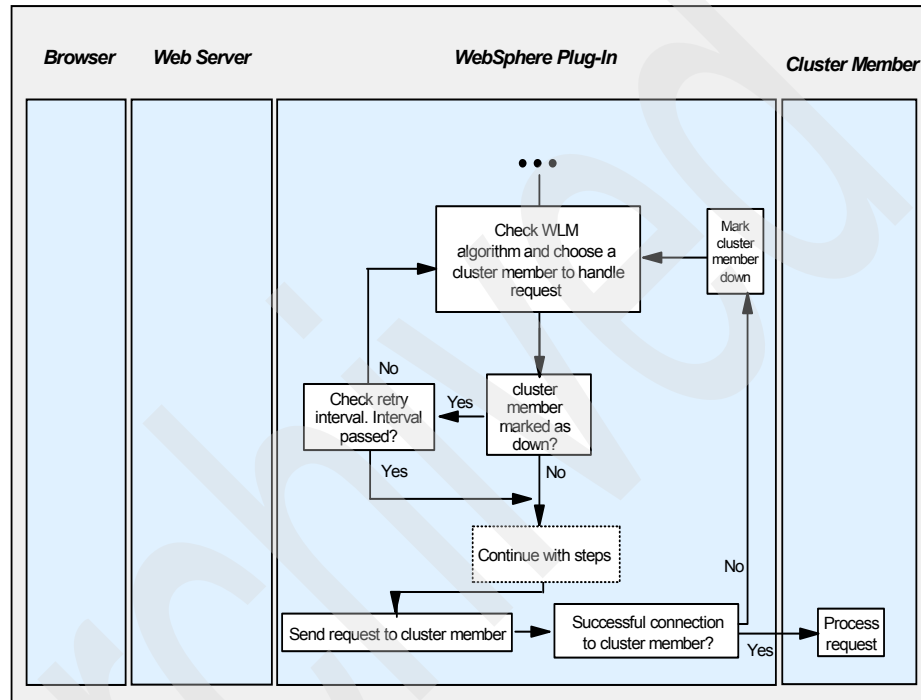


Figure 6-25 Failover selection process

For more information about failover, go to 6.10, “WebSphere plug-in behavior” on page 309. Most of the failure situations described here are discussed in detail in this section. That discussion includes logs and traces these errors lead to, for an administrator must be able to diagnose them correctly.

We also discuss primary and backup servers, an option within WebSphere Application Server to provide a second level of failover support.

6.7.1 Primary and backup servers

Starting with V5, WebSphere Application Server also implements a feature called primary and backup servers. When the plugin-cfg.xml is generated, all servers

are initially listed under the *PrimaryServers* tag, which is an ordered list of servers to which the plug-in can send requests.

There is also an optional tag called *BackupServers*. This is an ordered list of servers to which requests should only be sent if all servers specified in the Primary Servers list are unavailable.

Within the Primary Servers, the plug-in routes traffic according to server weight and/or session affinity. When all servers in the Primary Server list are unavailable, the plug-in will then route traffic to the first available server in the Backup Server list. If the first server in the Backup Server list is not available, the request is routed to the next server in the Backup Server list until no servers are left in the list or until a request is successfully sent and a response received from an application server. Weighted round robin routing is not performed for the servers in the Backup Server list.

Important: In WebSphere V6, the Primary and Backup Server lists are only used when the new partition ID logic is not used. In other words, when partition ID comes into play, then Primary/Backup Server logic no longer applies.

To learn about partition ID, please see “Session affinity” on page 281 and “Partition ID” on page 297.

To change an application server's role in a cluster, select **Servers -> Application servers -> <AppServer_Name> -> Web Server plug-in properties** and select the appropriate value from the Server Role pull-down menu (for example, change from Primary to Backup) as shown in Figure 6-26 on page 277.

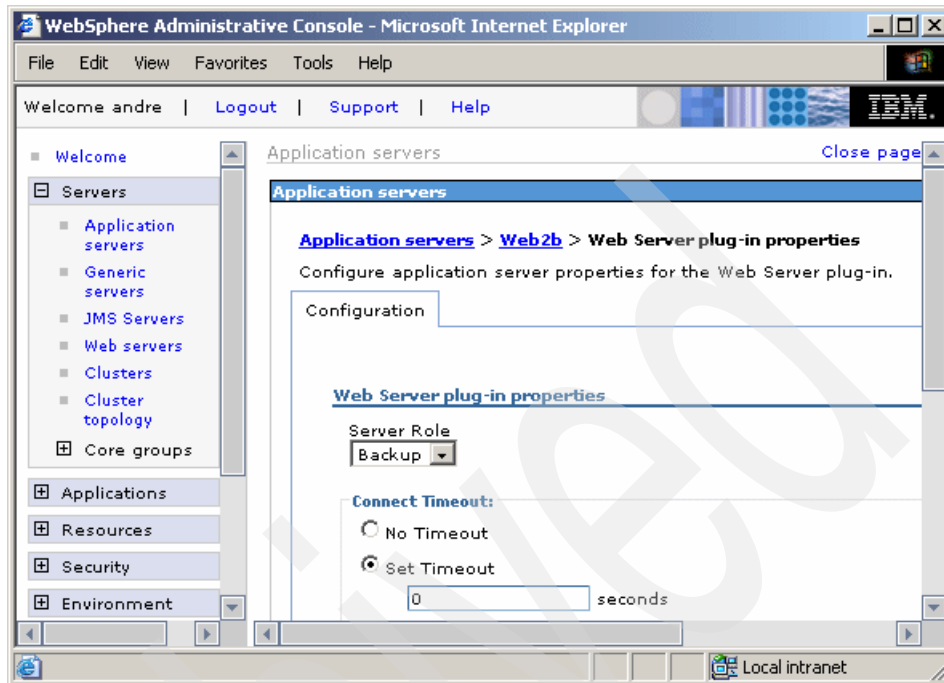


Figure 6-26 Setting up a backup server

All application server details in the plugin-cfg.xml file are listed under the `ServerCluster` tag. This includes the `PrimaryServers` and `BackupServers` tags, as illustrated in Example 6-5.

Example 6-5 *ServerCluster* element depicting primary and backup servers

```
...
<ServerCluster CloneSeparatorChange="false" LoadBalance="Round Robin"
Name="WEBcluster" PostSizeLimit="-1" RemoveSpecialHeaders="true"
RetryInterval="120">
  <Server CloneID="vve2m4fh" ConnectTimeout="5" ExtendedHandshake="false"
LoadBalanceWeight="2" MaxConnections="-1" Name="app1Node_Web1"
WaitForContinue="false">
    <Transport Hostname="app1.itso.ibm.com" Port="9080" Protocol="http"/>
  </Server>
  <Server CloneID="vv8kelbq" ConnectTimeout="5" ExtendedHandshake="false"
LoadBalanceWeight="2" MaxConnections="-1" Name="app2Node_Web2a"
WaitForContinue="false">
    <Transport Hostname="app2.itso.ibm.com" Port="9080" Protocol="http"/>
  </Server>
</ServerCluster>
```

```

    <Server CloneID="vv8keohs" ConnectTimeout="5" ExtendedHandshake="false"
    LoadBalanceWeight="2" MaxConnections="-1" Name="app2Node_Web2b"
    WaitForContinue="false">
      <Transport Hostname="app2.itso.ibm.com" Port="9081" Protocol="http"/>
    </Server>
    <PrimaryServers>
      <Server Name="app2Node_Web2a"/>
      <Server Name="app1Node_Web1"/>
    </PrimaryServers>
    <BackupServers>
      <Server Name="app2Node_Web2b"/>
    </BackupServers>
  </ServerCluster>
  ...

```

As mentioned before, the backup server list is only used when all primary servers are down. Figure 6-27 shows this process in detail.

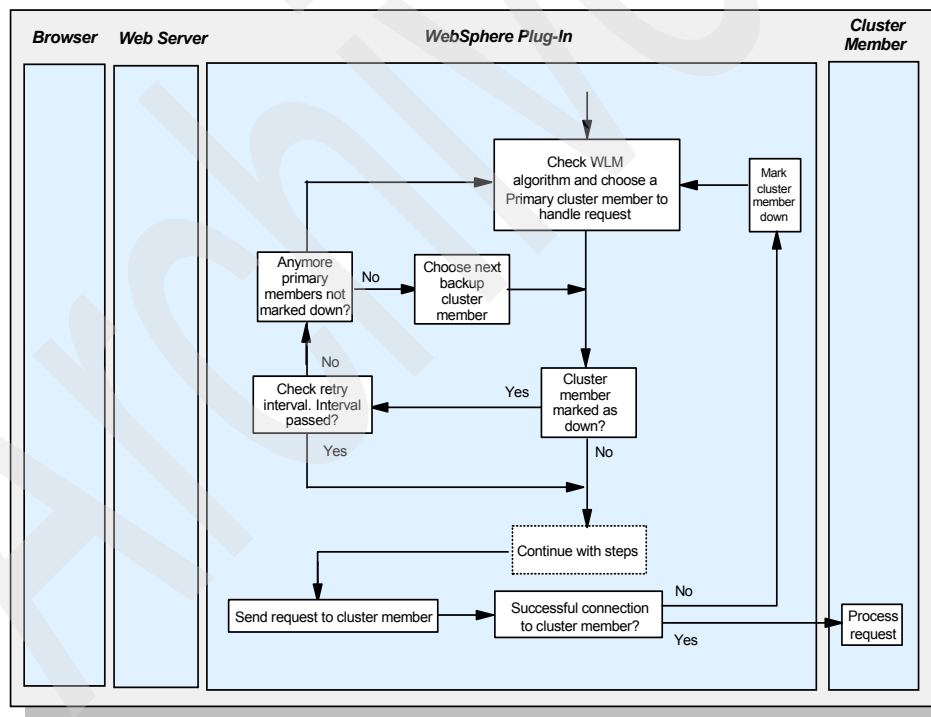


Figure 6-27 Primary and Backup server selection process

1. The request comes in and is sent to the plug-in.

2. The plug-in chooses the next primary server, checks whether the cluster member has been marked as down and the retry interval. If it has been marked as down and the retry timer is not 0, it continues to the next cluster member.
3. A stream is opened to the application server (if not already open) and a connection is attempted.
4. The connection to the cluster member fails. When a connection to a cluster member has failed, the process begins again.
5. The plug-in repeats steps 2, 3 and 4 until all primary servers are marked down.
6. When all primary servers are marked as down, the plug-in will then repeat steps 2 and 3 with the backup server list.
7. It performs steps 2 and 3 with a successful connection to the backup server. Data is sent to the Web container and is then returned to the user. If the connection is unsuccessful, the plug-in will then go through all the primary servers again and then through all the servers marked as down in the backup server list until it reaches a backup server that is not marked as down.
8. If another requests comes in and one of the primary server's retry timer is now at 0 the plug-in will try and connect to it.

6.8 HTTP session management

One of the other important functions that the plug-in provides, in addition to failover and workload management, is the ability to manage HTTP sessions.

In many Web applications, users move through the site based on a series of selections on pages they visit. Where the user goes next, and what the application displays as the user's next page (or next choice) may depend on what the user has chosen previously from the site. For example, if the user clicks the checkout button on a site, the next page must contain the user's shopping selections.

In order to do this, a Web application needs a mechanism to hold the user's state information over a period of time. However, HTTP alone does not recognize or maintain a user's state. HTTP treats each user request as a discrete, independent and stateless interaction.

The Java servlet specification proposes a mechanism for servlet applications to maintain a user's state information across multiple user hits. This mechanism, known as a session, allows a Web application developer to maintain all user

state information at the host, while exchanging only minimal information with a user's HTTP browser (mostly only session identification data).

Since the Servlet 2.3 specification, as implemented by WebSphere Application Server V5.0 and higher, only a single cluster member may control/access a given session at a time. After a session has been created, all following requests need to go to the same application server that created the session. This *session affinity* is provided by the plug-in. See 6.8.1, "Session affinity" on page 281 for more information.

If this application server is unavailable when the plug-in attempts to connect to it, the plug-in will choose another cluster member and attempt a connection. Once a connection to a cluster member is successful, the session manager will decide what to do with the session. The cluster member will find that it does not have the session cached locally and thus will create a new session.

To avoid the creation of a new session, a *distributed session* can be used to access sessions from other application servers.

There are two mechanisms to configure distributed sessions in WebSphere Application Server V6:

- ▶ Database persistence

Session state is persisted to a database shared between the clustered application servers. This feature was the only session persistence mechanism provided by earlier versions of WebSphere Application Server (pre-V5.0).

- ▶ Memory-to-memory replication, based on DRS (Data Replication Services), a feature that has been much simplified in WebSphere Application Server V6. It provides in-memory replication of session state between clustered application servers.

For information about how to configure distributed session management using either one of these mechanisms, please refer to 6.8.5, "Database session management configuration" on page 291 and 6.8.6, "Memory-to-memory replication configuration" on page 294.

The methods used to manage the sessions are very important in a workload-managed environment. There are different methods of identifying, maintaining affinity, and persisting the sessions; these are described in detail in the following sections.

6.8.1 Session affinity

As mentioned before, the Servlet 2.3 specification defines that, after a session has been created, all following requests need to go to the same application server that created the session.

However, in a clustered environment, there is more than one application server that can serve the client request. Therefore, the plug-in needs to read a request and be able to identify which cluster member should handle it.

Session identifiers are used to do this; they allow the plug-in to pick the correct cluster member and the Web container to retrieve the current session object.

Example 6-6 Example of a session identifier

JSESSIONID=0000A2MB4IJozU_VM8IffsMNfdR:v544d0o0

As shown in Example 6-6, the JSESSIONID cookie can be divided into four parts:

- ▶ Cache ID
- ▶ Session ID
- ▶ Separator
- ▶ Clone ID or Partition ID

Table 6-4 shows the mappings of these parts based on Example 6-6.

Table 6-4 JSESSIONID content

Content	Value used in the example
Cache ID	0000
Session ID	A2MB4IJozU_VM8IffsMNfdR
Separator	:
Clone ID (cluster member) - or - Partition ID	v544d0o0 a long numeric number

A clone ID is the ID of a cluster member, as shown in Example 6-7.

Example 6-7 Extract of plugin-cfg.xml showing the CloneID

```
<Server CloneID="vtnu14vu" ConnectTimeout="0" ExtendedHandshake="false"
LoadBalanceWeight="2" MaxConnections="-1" Name="app1Node_Web1"
WaitForContinue="false">
```

Important: There is a new processing logic in WebSphere V6 which is based on the so-called *partition ID*. JSESSIONID will include a partition ID instead of a clone ID when memory-to-memory replication in peer-to-peer mode is selected. Typically, the partition ID is a long numeric number.

For more information about partition ID, please refer to “Partition ID” on page 297.

6.8.2 Session identifiers

There are three methods of identifying a user's session to the application server. They are:

- ▶ Cookies
- ▶ URL rewriting
- ▶ SSL ID

This is a setting that can also be configured at the application server level, at the enterprise application level or at the Web module level (as explained in 6.8.4, “Session management configuration” on page 287).

Again, to learn more on this topic, please read Chapter 12, “Session management” of the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

Cookies

When this option is selected from the session manager pane, the plug-in will use the JSESSIONID to manage requests. This name is required by the Servlet 2.3 specification. The session management tracking mechanism can be performed at the application server, Web container, or the Web module level.

To use the cookie JSESSIONID, users must have their browsers set up to allow cookies.

None of the workload management issues discussed in “SSL ID” on page 284 applies to cookies. The browser can be connected to any Web server and there will be no effect on the session. Cookie session identifiers survive a Web server crash and, provided persistent sessions are enabled, also survive unavailability of the application server.

The session lifetime is governed by the cookie lifespan. By default, WebSphere defines its cookies to last until the browser is closed. It is also possible to define the maximum age of the cookie in the cookies configuration.

When to use cookies

Cookies are by far the most common method of tracking the session. They work well in a workload-managed environment, with ease-of-use advantages over SSL IDs.

They do not require additional application coding and can be used from static HTML links, unlike URL rewriting.

The fact that any user can turn off cookie support in his/her browser could be more important to your application. If this is the case, then one of the other options must be considered.

If security is important, then it is possible to use cookies in an SSL environment and not have the overhead of setting up SSL session ID management.

URL rewriting

URL rewriting (or URL encoding) is a useful mechanism that does not require users to enable cookies in their browsers, and yet still allows WebSphere to manage sessions.

The process of setting up URL rewriting, however, is not transparent to the Web application. It requires a developer to include specific commands to append the session information to the end of any HTTP link that will be used from the Web browser. The plug-in will search for any URL encoded session information about incoming requests and route them accordingly.

Rewriting the URLs can only be done on dynamic HTML that is generated within WebSphere, for example the output from JSPs or servlets:

```
<A HREF='<%=response.encodeURL("/store/catalog")%>'>Catalog</A>
```

Session information is lost if static HTML links are accessed, restricting the flow of site pages to dynamic pages only. From the first page, the user receives a session ID, and the Web site must continue using dynamic pages until the completion of the session.

There are no specific issues with using URL encoding in a workload-managed environment.

When to use URL encoding

Due to the restrictions mentioned above, the only situation in which URL encoding excels over the other options is when users have not enabled cookies in their browsers.

Because it is possible to select more than one mechanism to pass session IDs, it is also possible to compensate for users not using cookies. URL encoding could

be enabled and then used as a fallback mechanism if the users are not accepting cookies.

SSL ID

To use the SSL ID as the session modifier, clients need to be using an SSL connection to the Web server. This connection does not need to use client certificate authentication, but simply a normal server authentication connection. This can be enabled by turning on SSL support in the Web server. SSL configuration is described in the redbook *WebSphere Application Server V6: Security Handbook*, SG24-6316.

The session ID is generated from the SSL session information. This is passed to the Web server and then passed on to the plug-in. If more than one Web server is being used, then affinity must be maintained to the correct Web server, since the session ID is defined by the connection between browser and Web server. Connecting to another Web server will reset the SSL connection (a new session ID is generated).

SSL tracking is supported only for the IBM HTTP Server and Sun Java System Web server, Enterprise Edition.

It is possible to maintain the SSL session affinity using the Load Balancer from IBM WebSphere Edge Components. See 4.4.6, “SSL session ID” on page 114 for details.

SSL session ID cannot be used on its own in a clustered environment. It is not possible to add the cluster member ID to the end of the SSL session information, so another mechanism must be used. Either cookies or URL rewriting needs to be enabled to provide this function. The cookie or rewritten URL then contains session affinity information that enables the Web server to properly route requests back to the same server once the HTTP session has been created on a server. If cookies or URL rewriting are not enabled, then a session is created but there will be no mechanism to return the user to the correct cluster member at their next request.

The format of the cookie or URL rewrite is shown in Example 6-8.

Example 6-8 Affinity information format when using SSL

```
SSLJSESSION=0000SESSIONMANAGEMENTAFFINI:v544d0o0
```

This is the same format as described in Example 6-6 on page 281 but in place of the session ID is the word SESSIONMANAGEMENTAFFINI.

With SSL, the session timeout is not controlled by the application server. The session timeout delay is governed by the Web server and the Web browser. The lifetime of an SSL session ID can be controlled by configuration options in the Web server.

When to use SSL ID

When using a clustered environment, SSL ID requires a good deal of overhead to set up the infrastructure. There is also a single point of failure for each session: the Web server. If the Web server goes down, the user will lose the session ID and therefore access to session information.

SSL ID is also slower than other mechanisms, since the browser has to communicate using HTTPS and not HTTP. The inconsistency of browsers and their SSL handling could also affect how the application performs.

However, SSL provides a more secure mechanism of user identification for session information. The session identifier is difficult to copy.

If your Web site requires the highest security, then use SSL ID, but be aware that it comes with the overheads and deficiencies mentioned above. Consider using standard cookies over an SSL session instead.

6.8.3 Session management and failover inside the plug-in

As you know, the plug-in will always try and route a request that contains session information to the application server that processed the previous requests. If however the server that contains the session is not available to the plug-in when it forwards the request, then the plug-in can route the request to an alternate server. The alternate server can then retrieve the distributed session information according to the chosen distribution method (database or memory-to-memory replication).

Figure 6-28 on page 286 and the subsequent step-by-step explanation will help you to understand how the plug-in performs the failover.

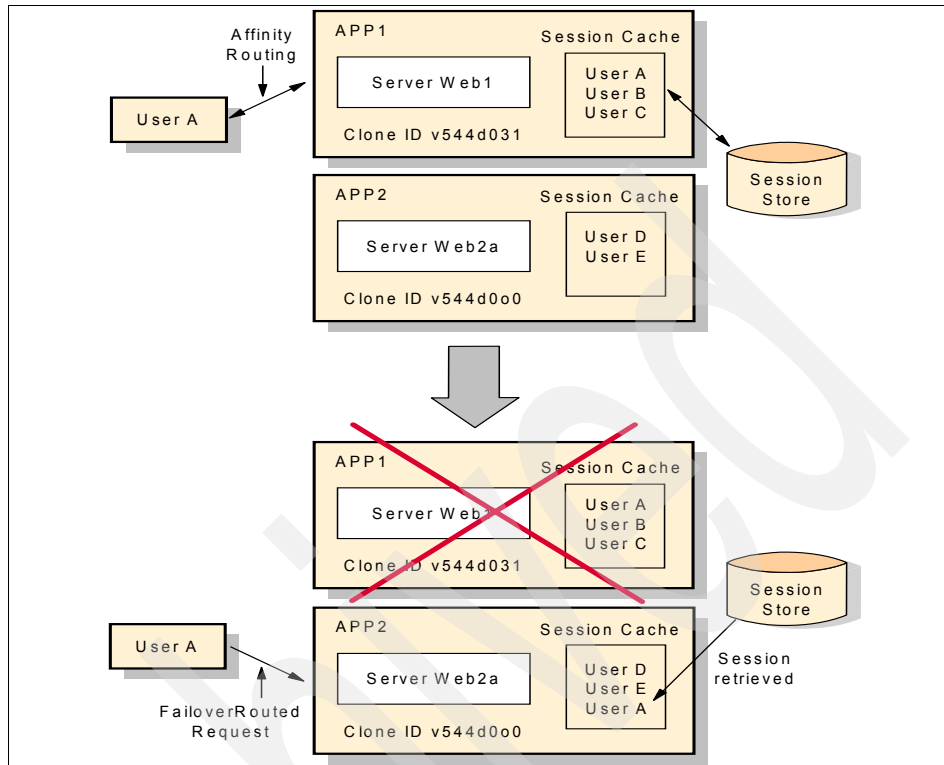


Figure 6-28 Session management example

Using Figure 6-28, the steps involved using our sample topology are:

1. The plug-in processes a request from user A to `http://OurWebServer/snoop`. The request also contains a JSESSION cookie with a session ID and CloneID of v544d031.
2. The plug-in matches the virtual host and URI to the cluster WEBcluster (composed by servers Web1 and Web2a, each one located in a different machine).
3. The plug-in then checks for session affinity and finds the CloneID of v544d031 in the request's JSESSIONID cookie.
4. It then searches for the CloneID of v544d031 in the plug-cfg.xml's list of primary servers and matches the CloneID to the Web1 application server.
5. The plug-in will then check to see if Web1 has been marked down. In our case it has not been marked down yet.
6. It then attempts to get a stream to Web1. Finding the server is not responding, Web1 is marked as down and the retry timer is started.

7. The plug-in then checks the session identifier again.
8. It then checks the servers. When it reaches Web1, it finds it is marked down and the retry timer is not 0, so it skips Web1 and checks the next cluster member in the primary list.
9. Web2a (CloneID v544d0o0) is selected and the plug-in attempts to get a stream to it. The plug-in either opens a stream or gets an existing one from the queue.
10. The request is sent and received successfully to Web2a (which retrieves the session information from the persistent session database or has it in-memory because of a previous replication) and sent back to user A.

The following sections explain the configuration steps needed to enable distributed session management using either a database or memory-to-memory replication.

6.8.4 Session management configuration

Session configuration settings are found in the Administrative Console in several different places. You need to set the General Properties for session management first, then you define the Distributed environment settings.

Basic session settings (General Properties)

In the General Properties configuration you define settings such as the Session tracking mechanism and the Session timeout. In WebSphere Application Server V6, session management can be defined at the following levels:

► Application server level

This is the default level. Session management configuration at this level is applied to all Web modules within the server. To define session management at the this level, select **Servers -> Application servers -> <AppServer_Name> -> Web Container Settings -> Session management**, as shown in Figure 6-29 on page 288.

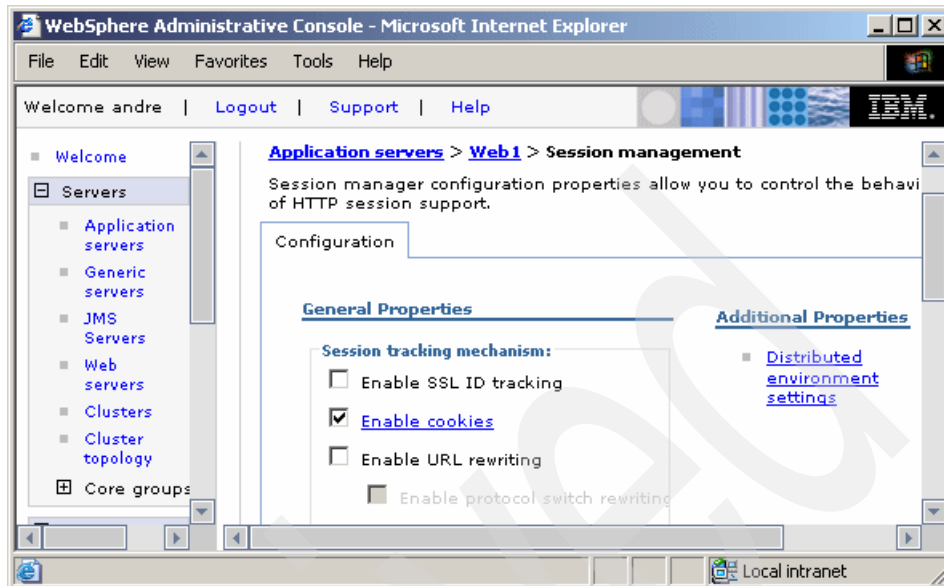


Figure 6-29 Session management at the application server level

► **Application level**

Configuration at this level is applied to all Web modules within the application. To override the inherited configuration (from the application server level) select **Applications -> Enterprise Applications -> <Application_Name> -> Session management**, check the **Override session management** option (as shown in Figure 6-30 on page 289) and click **Apply**.

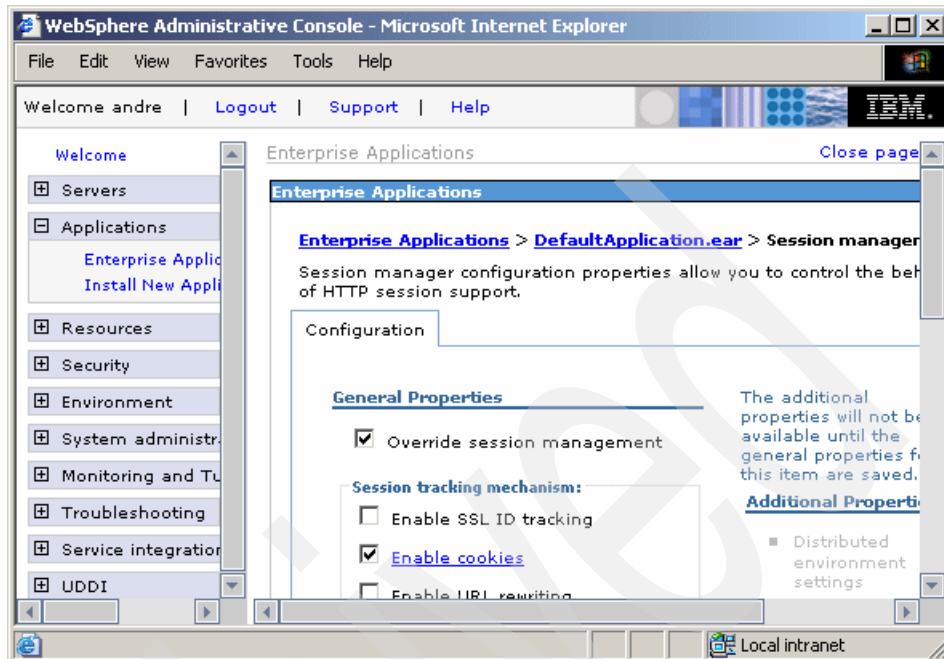


Figure 6-30 Overriding inherited session management configuration

► Web module level

Configuration at this level is applied only to the Web module. To override the inherited configuration (from the Application level), select **Applications -> Enterprise Applications -> <Application_Name> -> Web modules -> <Module_Name> -> Session Management**, check the **Override session management** option (similar to the one shown in Figure 6-30) and click **Apply**.

Distributed environment settings

The Distributed session settings define whether to use a database or memory-to-memory replication (the default is not to use distributed sessions) as well as tuning parameters (the write frequency).

Distributed sessions are also configured for the same three levels (Server, Application and Web module) as the General Properties:

► Application server level

This is also the default level applied to all Web modules within the server. To change it, select **Servers -> Application servers -> <AppServer_Name> -> Web Container Settings -> Session management -> Distributed environment settings** as shown in Figure 6-31 on page 290.

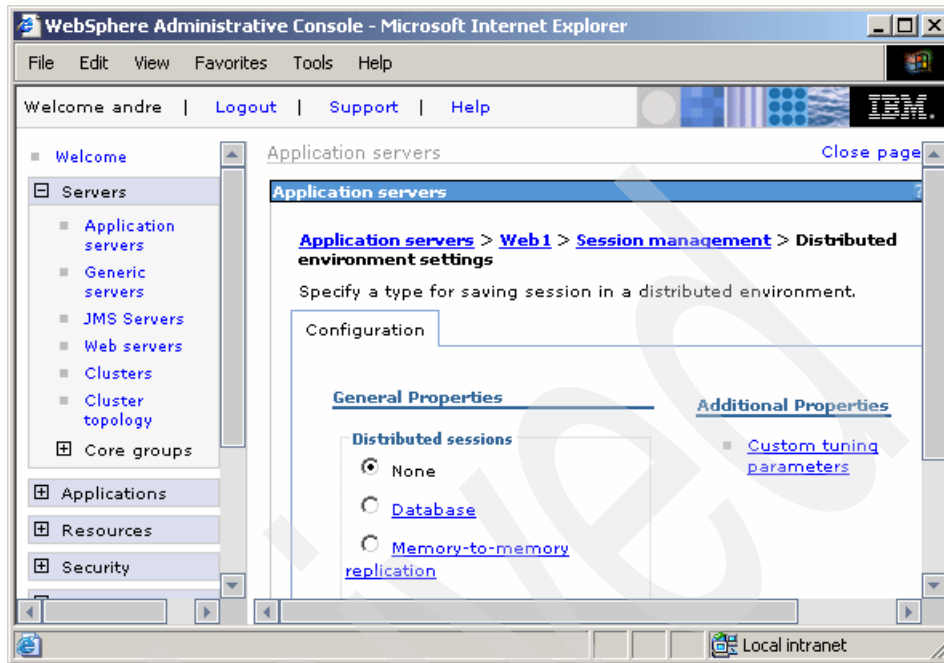


Figure 6-31 Configuring distributed sessions on the application server level

► Application level

To override the inherited configuration (from the Application Server level), select **Applications -> Enterprise Applications -> <Application_Name> -> Session management**, check the **Override session management** option (see Figure 6-30 on page 289) and click **Apply**.

This will enable the **Applications -> Enterprise Applications -> <Application_Name> -> Session management -> Distributed environment settings** option, where you can configure the same settings as for the application server level (see Figure 6-31).

► Web module level

To override the inherited configuration (from the Application Server level) select **Enterprise Applications -> <Application_Name> -> Web modules -> <Module_Name> -> Session management**, check the **Override session management** option (similar to the one in Figure 6-30 on page 289) and click **Apply**.

This will enable the **Enterprise Applications -> <Application_Name> -> Web modules -> <Module_Name> -> Session management -> Distributed environment settings** option, where you also can configure the same settings as for the application level.

Please refer also to Chapter 12, “Session management”, of the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451, where you can find many more details on how to configure sessions.

6.8.5 Database session management configuration

The use of a persistent store for sessions is not limited to use in a failover situation. It can also be used when an administrator requires greater control over the session cache memory usage.

In this chapter, we cover the basic setup and settings that are important to workload management and failover. Chapter 12, “Session management”, of the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451, gives you additional information.

The two steps needed to set up database session persistence are:

- ▶ Creating a session persistence database on the database server
- ▶ Enabling database persistent session management for each application server

Creating a session persistence database

Follow these steps to enable persistent sessions using a DB2 database:

1. Create a database that can be used for session management. The session manager performs best when it has exclusive use of the database and its connections. It is possible to use an existing database, but it is best to create a new one. For this example, we are using DB2.

Open a DB2 command window and type:

```
db2 create db sessdb
```

2. If you are using multiple nodes and a multi-member, clustered environment, then this database needs to be cataloged at each physical machine.
 - a. First, we need to catalog the DB2 node that contains the session database.

If the node is not already cataloged, open a command window and type:

```
db2 catalog tcpip node <node_name> remote <remote_hostname> server  
<service_name>
```

where:

- <node_name> is an arbitrary name for identification, for example sessnode.

- <remote_hostname> is the host name that DB2 can use to connect to the DB2 server containing the session database, for example sesshost.
 - <service_name> is the port number that DB2 is listening to on the remote DB2 server. The default is 50000.
- b. Next, catalog the database at that node using the command:
- ```
db2 catalog db sessdb as sessdb at node <node_name>
```
- Where, <node\_name> is the node name specified in the previous step.
- c. Verify that it is possible to connect to this newly cataloged database. Type the command:
- ```
db2 connect to sessdb user <userid> using <password>
```
3. Set up a DB2 JDBC provider using the WebSphere administration tools, if you have not done so already.
4. Set up the data source for use with the session manager by using the WebSphere administration tools. Within the JDBC provider that you have just created, configure a new data source.
- For information about how to set up JDBC providers and data sources, refer to 7.2, “JDBC resources”, of the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

Tip: Even though JDBC providers and JDBC data sources can be configured at cell, node, cluster, or server levels, they are instantiated only within an application server. In other words, a cell level data source is instantiated by every application server at every node.

For most Network Deployment environments, creating the mentioned JDBC provider and JDBC data source at cell level or cluster level is an excellent idea that will save a lot of time. However, care must be taken when defining local paths to libraries of the JDBC provider: the best trick is to use node-level defined WebSphere variables.

It is still required to catalog the DB2 databases at each node's DB2 client, no matter how the JDBC providers and JDBC datasources are configured.

Enabling database persistent session management

To enable persistent sessions, repeat the following steps for each application server in the cluster:

1. Click **Servers -> Application servers -> <AppServer_Name> -> Web Container Settings -> Session management -> Distributed environment settings**.

2. Select the **Database** radio button. This takes you automatically to the Database settings panel shown in Figure 6-32.

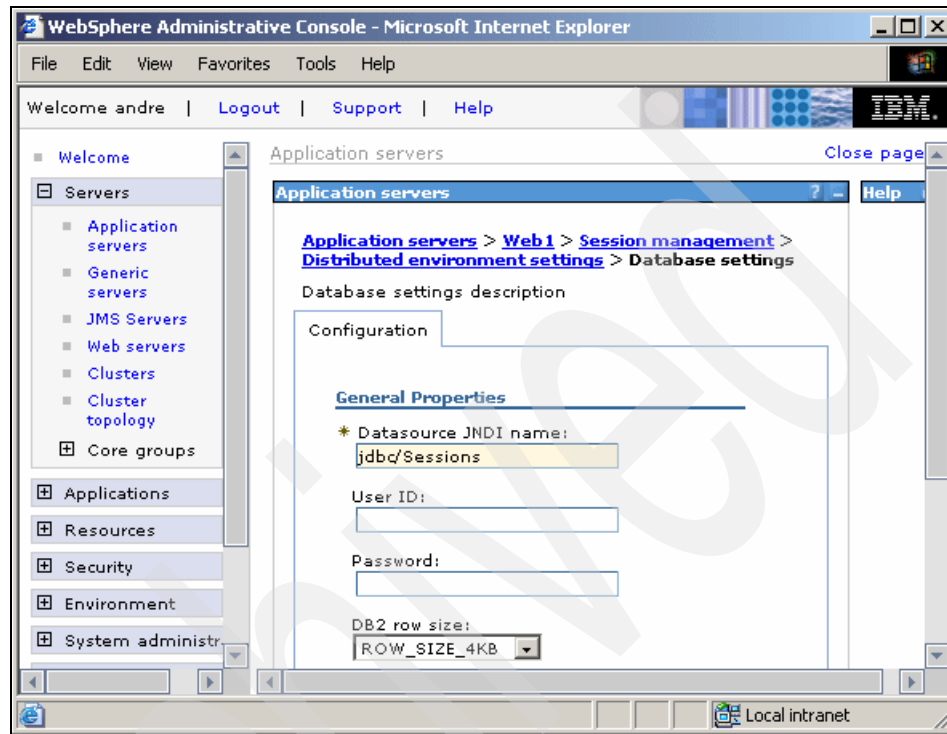


Figure 6-32 Database settings for the session manager

3. Enter values where necessary for the configuration properties:
 - Enter your Datasource JNDI name. The data source must be a non-JTA enabled data source.
 - If required, enter a user ID and password.
 - If you are using DB2 and you anticipate requiring row sizes greater than 4 KB, select the appropriate value from the DB2 row size pull-down. See 12.9.5, “Larger DB2 page sizes and database persistence”, of the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451, for more information about this setting.
 - If you intend to use a multi-row schema, check the appropriate box. Again, information about multi-row schemes can be found in section 12.9.6, “Single and multi-row schemas (database persistence)” of the *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

4. Click **OK**.
5. Repeat these configuration steps for all cluster members.

How database session persistence works

Upon the first creation of a session, you should see that a table called Session is created for you. The cluster members will now make use of this session database.

In our example, we have two cluster members in WEBcluster: Web1 and Web2a. If we make an initial session-based request to Web1 and shut it down afterwards, the plug-in will then try to route requests through to Web2a. In general, if the plug-in is unable to contact the initial server, it will try another cluster member. The next cluster member that responds will check its local cache and, upon not finding the session, will connect to the session database and retrieve the session from there.

When the servlet returns the result to the plug-in, the cluster member that served the page is appended to the list of clone IDs in the session identifier. The session identifiers will now look like those shown in Example 6-9.

Example 6-9 Example of session JSESSION cookies with multiple clone IDs

JSESSIONID=0002VIS4-cfZD1hkdxF7MKGX5XZ:vu6kr6r0:vu6krbkq

The plug-in now tries both of these servers before resorting to the workload management algorithm again. Once the original server is available again, all session requests will return to it.

This behavior means that the plug-in performs faster and the user will go to a server that has a local cached version of the session.

If you wish to test whether the failover and persistent sessions are working, follow the steps in 6.10, “WebSphere plug-in behavior” on page 309.

6.8.6 Memory-to-memory replication configuration

Memory-to-memory replication enables the sharing of sessions between application servers without using a database. Memory-to-memory replication is based on the Data Replication Services (DRS) of WebSphere Application Server V6 which allows the definition of replication domains whose members can have their HTTP sessions replicated among themselves.

The benefits of using memory-to-memory replication rather than a database for session persistence is that the overhead and cost of setting up and maintaining a real-time, production database, such as preparing a machine, installing and

configuring a database, starting and so on, is not needed. Also, the database becomes a SPOF (Single Point Of Failure) for session persistence and certain cost and effort is required to solve this issue at the database level. However, it was found that database persistence might perform better than a badly configured memory-to-memory replication. See 1.1.5, “Session state” on page 10 for additional information.

All features available in database persistence are available in memory-to-memory replication as well, except for DB2 variable row size and multi-row features, which are features specific to a database.

To effectively deploy memory-to-memory replication in clustered environments, especially large ones, implementers must think carefully about how to exactly configure the replicator topology and settings. If care is not taken then the amount of memory and resource usage taken by session replication can increase significantly.

See 6.8.7, “Understanding DRS (Data Replication Services)” on page 297 for information about how the underlying mechanism for memory-to-memory replication works.

Configuring memory-to-memory session management

The following is a high-level description of how to set up this function.

Refer to 8.5, “Installing WebSphere and configuring clusters” on page 395 or, again, to Chapter 12 of the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451, for details on how to configure clusters and memory-to-memory replication.

1. Create a replication domain (if not already done so during cluster creation).

Replication domains define the set of replicator processes that communicate with each other through DRS (Data Replication Services). The number of replicas to be created for each HTTP session can also be configured. Usually, there will be a default replication domain for each created cluster (by selecting the non-default option **Create a replication domain for this cluster** (click **Clusters -> New**). If you did not create a replication domain during the initial cluster definition, you can always do it later by selecting **Environment -> Replication domains -> New**.

2. Configure each cluster members’ replication properties by selecting **Servers -> Application servers -> <AppServer_Name> -> Web Container Settings -> Session management -> Distributed environment settings -> Memory-to-memory replication** (as shown in Figure 6-33 on page 296).

Here you can define the Replication domain as well as the Replication mode.

3. Restart the cluster.

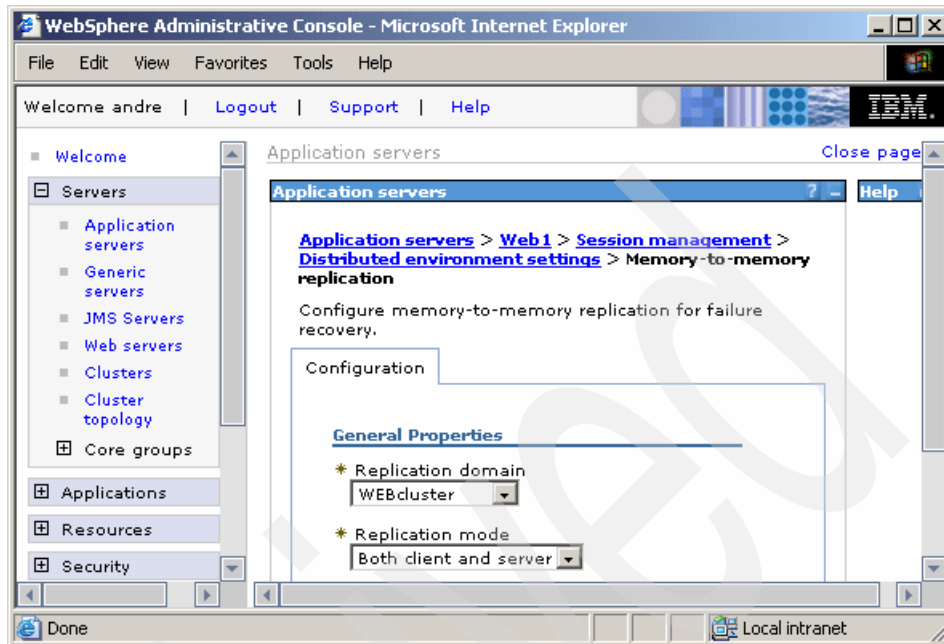


Figure 6-33 Memory-to-memory replication of HTTP sessions

How memory-to-memory session replication works

In our example, we have two cluster members in WEBcluster: Web1 and Web2a. If we make an initial session-based request to Web1 and shut it down afterwards, the plug-in will then try and route requests through to Web2a. In general, if the plug-in is unable to contact the initial server, it will try another cluster member. The next cluster member that responds will already have the session in memory since it has been replicated using memory-to-memory replication.

Again, when the servlet returns the result to the plug-in, the cluster member that served the page is appended to the list of clone IDs in the session identifier. The session identifiers now look like those shown in Example 6-10.

Example 6-10 Example of session JSESSION cookies with multiple clone IDs

JSESSIONID=0002YBY-wJPYdXZvCZkc-LkoBBH:v544d031:v544d0o0

The plug-in now tries both of these servers before resorting to the workload management algorithm again. Once the original server is available again, all session requests will return to it.

If you wish to test whether the failover and persistent sessions are working, follow the steps in 6.10, “WebSphere plug-in behavior” on page 309.

Important: Depending on your memory-to-memory session replication configuration, this behavior might be slightly different. See the section Partition ID below for details.

Partition ID

As mentioned earlier, there is a new logic in WebSphere V6 called the *partition ID*, which plays a role when memory-to-memory session replication is configured in peer-to-peer (or “Both”) mode (see “Data replication service topologies” on page 299 for information). The partition ID is related to the WebSphere HAManager (described in Chapter 9, “WebSphere HAManager” on page 465).

For every cluster member, the HAManager generates a corresponding partition ID and maintains a table where the partition ID is mapped to the clone ID. This mapping is sent to the plug-in via internal headers as part of the response to the client request. In every response, the plug-in looks for the two headers that hold the partition table and the table version. If the partition ID table is sent as part of the response, the mapping table is received and stored in memory by the plug-in. The headers are then removed and are not sent to the client.

When JSESSIONID is received, the plug-in checks if the IDs match any of the clone IDs. If there is a match, the corresponding cluster member is used. If there is no match *and* if the partition ID table was previously received by the plug-in, then IDs are checked for known partition IDs. If there is a match for a partition ID, the corresponding clone ID is used for the request. If there is no match, existing processing logic, as described before, continues.

If the plug-in uses the partition ID table to send a request but the cluster member is down, the plug-in sends a separate request to retrieve the latest partition ID table. The new table is then used for handling the request.

This new logic helps in hot session fail over. The plug-in is now made aware of which server to fail over for a given cluster member. The partition ID table has the required mapping of partition ID to clone ID and is sent to the plug-in by the session management, if the version sent by the plug-in is not the latest copy. Example 6-19 on page 328 is a trace that shows the partition ID failover behavior.

6.8.7 Understanding DRS (Data Replication Services)

As mentioned in 6.8.6, “Memory-to-memory replication configuration” on page 294, this session replication mechanism uses the Data Replication Service (DRS) to replicate data between multiple application servers in a cluster without using a database. Using this method, sessions are stored in the memory of two or more application servers, providing the same functionality as a database for

session persistence. Separate threads handle this functionality within an existing application server process.

What is DRS?

Data replication service (DRS) is an internal WebSphere Application Server component designed for generic data replication. Apart from the Session manager, it is also used to replicate dynamic cache data and stateful session beans across many application servers in a cluster.

The advantages of using this method of session persistence are:

- ▶ Flexible configuration options such as peer-peer and client/server.
- ▶ Eliminates the overhead and cost of setting up and maintaining a real-time production database.
- ▶ Eliminates the single point of failure problem that can occur when using a database.

DRS has been greatly simplified from WebSphere Application Server V5.x to V6. Replicators and partitions are not part of the configuration anymore, and thus these terms have been abandoned.

Replication domain

The memory-to-memory replication function is accomplished by the creation of a data replication service instance in an application server that communicates to other data replication service instances in other application servers. You must configure this data replication service instance as a part of a replication domain.

A new replication domain can be created at any time using the WebSphere Administrative Console, but usually you will create one while creating a new application server cluster, as explained in the example in 8.5.8, “Configure distributed session management” on page 411.

You can view (and change) a replication domain configuration by selecting **Environment -> Replication domains -> <Replication_Domain_Name>**, as shown in Figure 6-34 on page 299.

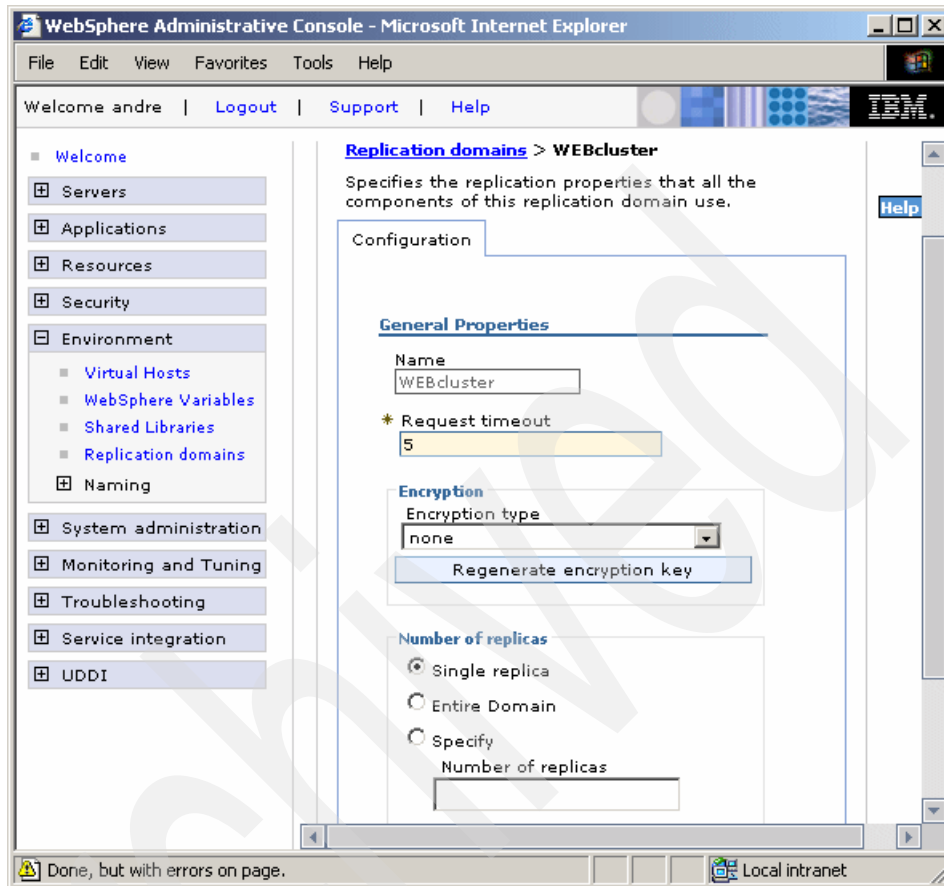


Figure 6-34 Replication domain configuration

The most important setting is the *number of replicas* created for every entry or piece of data that is replicated in the replication domain. In our case, it is the number of copies of HTTP session data that will be replicated to other application servers.

Data replication service topologies

Servers dynamically join a replication domain according to their configuration. Whenever you configure memory-to-memory replication for an application server, you are asked for a replication domain, as can be seen in Figure 6-33 on page 296.

As explained before, the memory-to-memory replication function is accomplished by the creation of a data replication service instance in an application server that

communicates to other data replication service instances in remote application servers.

There are three possible modes you can set up in which a replication service instance can run:

- ▶ **Server mode:** In this mode, a server only stores backup copies of other application server sessions. It does not send out copies of sessions created in that particular server.
- ▶ **Client mode:** In this mode, a server only broadcasts or sends out copies of the sessions it owns. It does not receive backup copies of sessions from other servers.
- ▶ **Both mode:** In this mode, the server simultaneously sends out copies of the sessions it owns and acts as a backup table for sessions owned by other application servers. Sometimes this also referred to as peer-to-peer mode.

You can select the replication mode of server, client, or both when configuring the session management facility for memory-to-memory replication. The default is both.

This application server setting and the number of replicas of the replication domain together allow the definition of several possible replication topologies. You can find much more information about this subject in section 12.9.2, “Memory-to-memory replication” of the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

6.8.8 Session management tuning

Performance tuning for session management consists of defining the following:

- ▶ How often session data is written (write frequency settings).
- ▶ How much data is written (write contents settings).
- ▶ When the invalid sessions are cleaned up (session cleanup settings).

Writing frequency settings

You can select from three different settings that determine how often session data is written to the persistent data store:

- ▶ **Time-based:** The session data will be written to the persistent store based on the specified write interval value.
- ▶ **End of servlet service:** If the session data has changed, it will be written to the persistent store after the servlet finishes processing an HTTP request.
- ▶ **Manual update:** The session data will be written to the persistent store when the `sync()` method is called on the `IBMSession` object.

Note: The last access time attribute is always updated each time the session is accessed by the servlet or JSP whether or not the session is changed. This is done to make sure the session does not time out.

- ▶ If you choose the end of servlet service option, each servlet or JSP access will result in a corresponding persistent store update of the last access time.
- ▶ If you select the manual update option, the update of the last access time in the persistent store occurs on sync() call or at later time.
- ▶ If you use time-based updates, the changes are accumulated and written in a single transaction. This can significantly reduce the amount of I/O to the persistent store.

To configure the session update frequency, you must select **Servers -> Application servers -> <AppServer_Name> -> Web Container Settings -> Session management -> Distributed environment settings -> Custom tuning parameters**, as shown in Figure 6-35 on page 302.

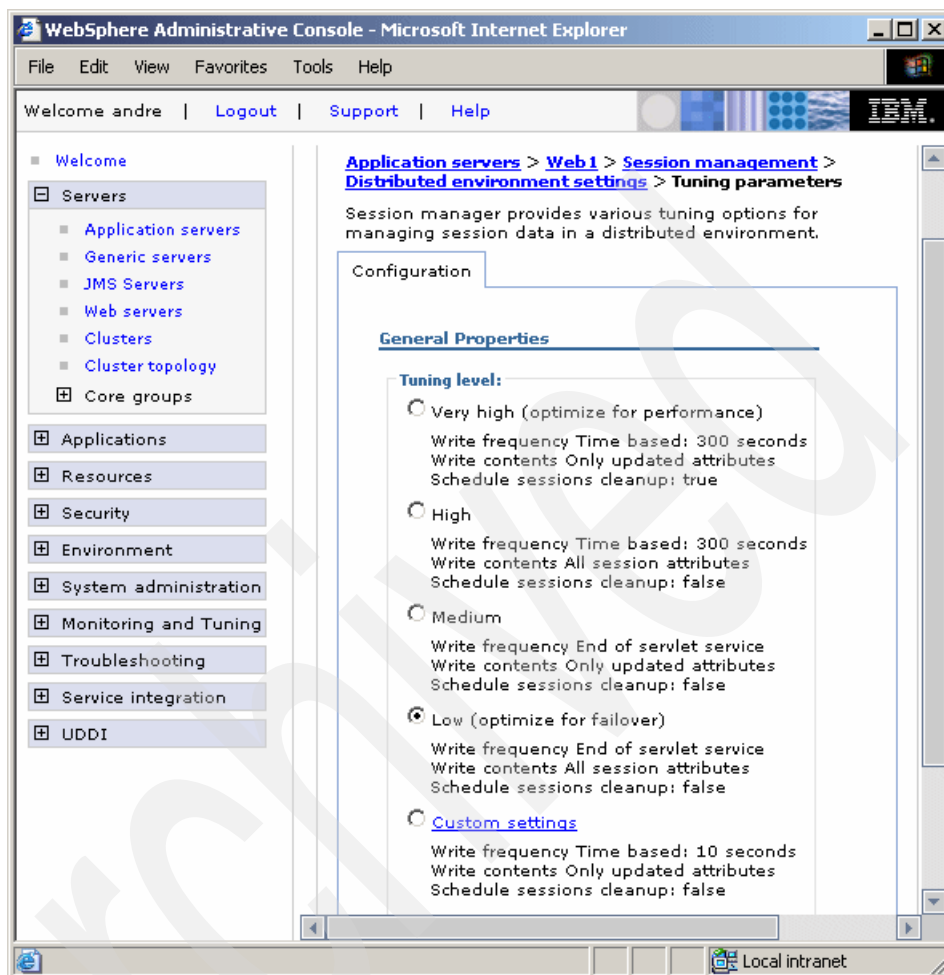


Figure 6-35 Session update tuning

Time-based (specified time interval)

In this configuration, session data is written into the database (or published into the replication domain) at preset time intervals. Any modified session data since the last update is lost during a failover.

In Figure 6-35, you can see the choices for some preset time intervals. A custom interval can be set by selecting **Custom settings**, as shown in Figure 6-36 on page 303.

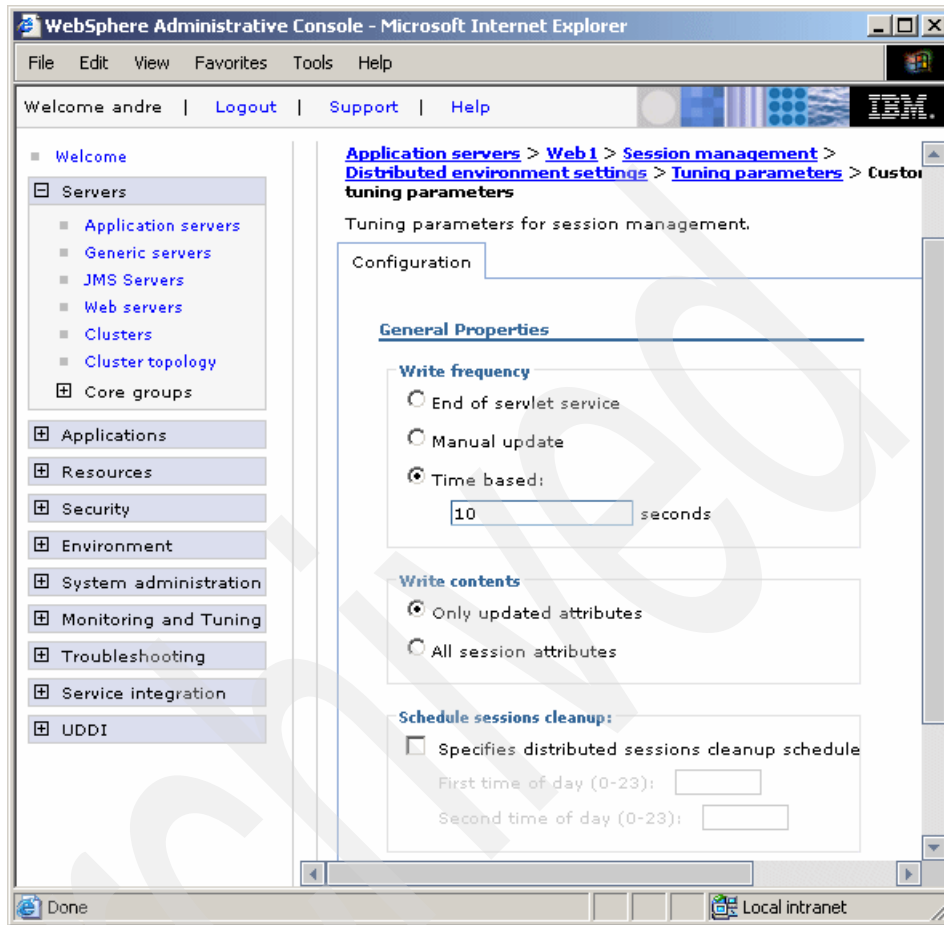


Figure 6-36 Custom settings for session update

End of servlet service

When the write frequency is set to the end of servlet service option, WebSphere writes the session data to the persistent store at the completion of the `HttpServlet.service()` method call. Exactly what is written depends on the write content settings.

To choose the **End of servlet service** setting, you can either select a tuning level of **Medium** or **Low** or select **Custom**, if you wish to change the Schedule sessions cleanup setting. See Figure 6-36.

Manual update

In manual update mode, the session manager only sends changes to the persistent data store if the application explicitly requests a save of the session information.

Note: Manual updates use an IBM extension to HttpSession that is not part of the Servlet 2.4 API. Such source code is not portable to other application servers.

Manual update mode requires that an application developer use the IBMSession class for managing sessions. When the application invokes the `sync()` method, the session manager writes the modified session data and last access time to the persistent store. The session data that is written out to the persistent store is controlled by the write contents option selected.

To choose the Manual update setting you must opt for the Custom tuning level and select **Manual update** as the write frequency. Again, refer to Figure 6-36 on page 303.

For more information about Manual update, please see section 12.9.3, “Session management tuning” of the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

6.9 Troubleshooting the Web server plug-in

If problems occur with the plug-in, you can use the logging and tracing features of WebSphere to find a solution.

6.9.1 Logging

Logging for the plug-in can occur in three different places:

1. The log specified in the plug-in configuration file (the default is `http_plugin.log`).
2. WebSphere Application Server's activity.log.
3. Web server error log.

The plug-in log

The plug-in log configuration can be set for each Web server by selecting **Servers -> Web Servers -> <WebServer_Name> -> Plug-in properties:**

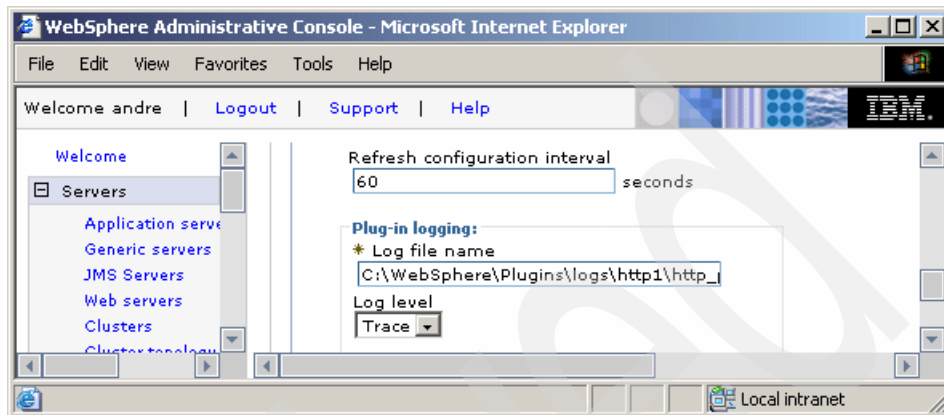


Figure 6-37 Plug-in log configuration

The LogLevel can be set to:

- ▶ Trace (see "Trace" on page 308)
- ▶ Stats (see "LogLevel Stats" on page 306)
- ▶ Warn
- ▶ Error

Error contains the least information and Trace contains the most. If you select Trace, then there is much information created for each access to the Web server. This setting should not be used in a normally functioning environment, because the file will rapidly grow and there will be a significant reduction in performance. To learn more about Trace, see 6.9.2, "Trace" on page 308.

The name of the plug-in log file in a remote Web server is, by default, `<PLUGIN_HOME>/logs/<yourWebServer>/http_plugin.log`, but this can be changed in the Log file name field of the Plug-in properties (see Figure 6-37). Remember that each Web server plug-in will write its own plug-in log file in its local file system.

If you wish to change the location of the log, change the name value and a new log will be created and used the next time the plug-in refreshes its configuration.

Every time the Web server starts, it will create the plug-in log if it does not exist. This is a simple way to see if the plug-in is running. If there is no log created when the Web server is started, then either the plug-in was not initialized or it was unable to create or open the log. Make sure that wherever you place the log, the Web server has the authority to create and modify files.

If there is a problem with settings, creation, or modification of the log, an error will appear in the Web server log.

Successful startup will be signified by lines similar to those shown in Example 6-11 appearing in the log.

Example 6-11 Message shown on a successful startup on Windows

```
.
.
(...)PLUGIN: Plugins loaded.
(...)PLUGIN: -----System Information-----
(...)PLUGIN: Bld version: 6.0.0
(...)PLUGIN: Bld date: Oct 1 2004, 22:03:36
(...)PLUGIN: Webserver: IBM_HTTP_Server/6.0 Apache/2.0.47 (Win32)
(...)PLUGIN: Hostname = http1
(...)PLUGIN: OS version 5.0, build 2195, 'Service Pack 4'
(...)PLUGIN: -----
.
.
```

LogLevel Stats

Since WebSphere Application Server V5.1 the plug-in has been enhanced to support a new log level called *Stats*. When this log level is selected, it will list all ERROR and WARNING messages as well as additional server usage statistics and server status used for server selection, yet with less verbosity than the Trace level. For every request, the two messages shown in Example 6-12 are logged.

Example 6-12 Messages for Stats Loglevel

```
[Fri Jan 21 16:34:08 2005] 00000af0 00000b34 - STATS: ws_server_group:
serverGroupCheckServerStatus: Checking status of app1Node_Web1, ignoreWeights
1, markedDown 0, retryNow 0, wlbAllows -1 reachedMaxConnectionsLimit 0
[Fri Jan 21 16:34:08 2005] 00000af0 00000b34 - STATS: ws_server:
serverSetFailoverStatus: Server app1Node_Web1 : pendingConnections 0
failedConnections 0 affinityConnections 2 totalConnections 3.
```

There will be more than one occurrence of the serverGroupCheckServerStatus message if the server was not selectable for the current request.

The following server status attributes are used to decide whether the server can handle the request (see Example 6-12 on page 306):

- ▶ **ignoreWeights:** This indicates whether the current request should ignore weights. This will be set to 1 if the request has an associated affinity server or if load balancing is chosen to be random.
- ▶ **markedDown:** This indicates whether the server is currently marked down and can't be selected for the current request.
- ▶ **retryNow:** This indicates that the marked down server can be retried now.
- ▶ **wlbAllows:** This indicates whether the current server has positive weights. A server can be selected only if it has positive weights when weights can't be ignored (ignoreWeights set to 0).
- ▶ **reachedMaxConnectionsLimit:** This indicates whether the current server has reached maximum pending connections. A server won't be selected if this attribute has a value of 1.

The following are the additional server statistics. This information is logged after getting the response from the server for the current request. Note that this is collected per process.

- ▶ **pendingConnections:** Number of requests for which a response is yet to be received from the server.
- ▶ **failedConnections:** Number of failed requests to the server.
- ▶ **affinityConnections:** Number of requests that are going to an affinity server.
- ▶ **totalConnections:** Total number of requests that are handled by the server.

WebSphere Application Servers activity.log

The activity.log contains errors that occur when:

1. Generation of the plug-in configuration is launched automatically or manually from the Administrative Console and an error occurs.

When launching the generation of the plug-in configuration and something goes wrong, the errors are listed in the activity.log file. The exception to this is if it is run from the command line. In this case, all errors are written to the screen.

2. An error occurs when the Web container attempts to process a request passed to it by the plug-in. For example, where the information that is in the plugin-cfg.xml does not map to the information in the configuration repository.

If the plug-in configuration is changed manually (which is not recommended), it can lead to inconsistencies with the administrative repository. For example if you manually added a virtual host to the definitions in the plugin-cfg.xml file but did not add the same virtual host to the configuration repository. The error message would be:

```
PLGN0021E: Servlet Request Processor Exception: Virtual Host/WebGroup  
Not Found: The host hostname has not been defined.
```

Web server error log

Errors show up in the error log of the Web server when the plug-in has been unable to access the plug-in log. With IBM HTTP Server, the log is error_log on AIX and error.log on Windows. It is in the logs directory under the HTTP Server installation directory.

A similar error will occur if the native.log file is locked or inaccessible to the Web server. By default, all users have read, write, and executable permissions for the WebSphere logs directory. This means that there should be no problems with access unless the log location is changed from the default.

6.9.2 Trace

There are two areas of the plug-in configuration where tracing can be enabled: the actions of the plug-in itself, and the generation of the configuration file.

Tracing the plug-in

Tracing is enabled in the plug-in by setting the Log level to Trace, as specified in 6.9.1, “Logging” on page 304.

A good method for starting a new trace is to change the LogLevel to Trace and change the name of the file to a new value, for example traceplugin.log. This way, you know where your trace starts and you have an easily manageable trace file.

Once you are finished, you can return the plug-in configuration file to its original settings and do not need to stop the Web server or the application server for any of this to work.

The trace itself is quite straightforward to follow. If you are tracing requests, you will see a request handled as explained in Figure 6-23 on page 265.

Tracing provides you with information about:

- The virtual hosts the plug-in is using. You can view how the virtual host is matched to the request. One of the most common problems is specifying the

same alias in multiple virtual hosts and so not giving the plug-in a unique route to follow.

- ▶ The URIs the plug-in is searching. If you are unsure as to why a request is not reaching your application, use a trace to see how the plug-in is matching your browser request.
- ▶ Workload management selection process. Tracing is useful if you wish to validate how your cluster members are being selected and watch failover mechanisms operate.
- ▶ Observing the transport mechanism being used and connections or failures in connection.
- ▶ Session management.

6.10 WebSphere plug-in behavior

There are various scenarios that can occur in the event of a failure somewhere in your system. This section examines some example scenarios and shows the system behavior.

The tests are designed to access a cluster from a browser via the plug-in in a Web server. The number of nodes, cluster members, and physical servers is specified in the two areas of investigation: Normal operation and Failover operation.

Each of the test setup procedures assumes that the sample application (DefaultApplication.ear) and the BeenThere application have already been installed and that there has been a successful connection to:

- ▶ `http://yourWebserver/snoop`
- ▶ `http://yourWebserver/hitcount`
- ▶ `http://yourWebserver/wlm/BeenThere`

Refer to 8.7, “Installing and configuring BeenThere” on page 426 for information about how to obtain, install, and configure BeenThere.

It is also assumed that a persistent session database or memory-to-memory replication has been configured to handle distributed sessions. See 6.8.3, “Session management and failover inside the plug-in” on page 285 for details.

6.10.1 Normal operation

In this section, we discuss using the WebSphere samples to show how the plug-in achieves workload management and session affinity. These tests were performed in our lab, using the sample topology described in Chapter 8, “Implementing the sample topology” on page 387.

We are interested in how the Web server plug-in distributes browser requests to the available WebSphere Web containers. We are sending our requests only to the Web server http1 (not through the Load Balancer). This way, we only need to observe the plug-in behavior on a single Web server.

Workload management with the plug-in

To check the behavior of plug-in workload management:

1. Use the default settings in the plugin-cfg.xml file.
2. Open a browser and go to the URL:

`http://http1/wlm/BeenThere`

where *http1* is the host name of our Web server in the sample topology.

3. Observe the Server field as shown in Figure 6-38.

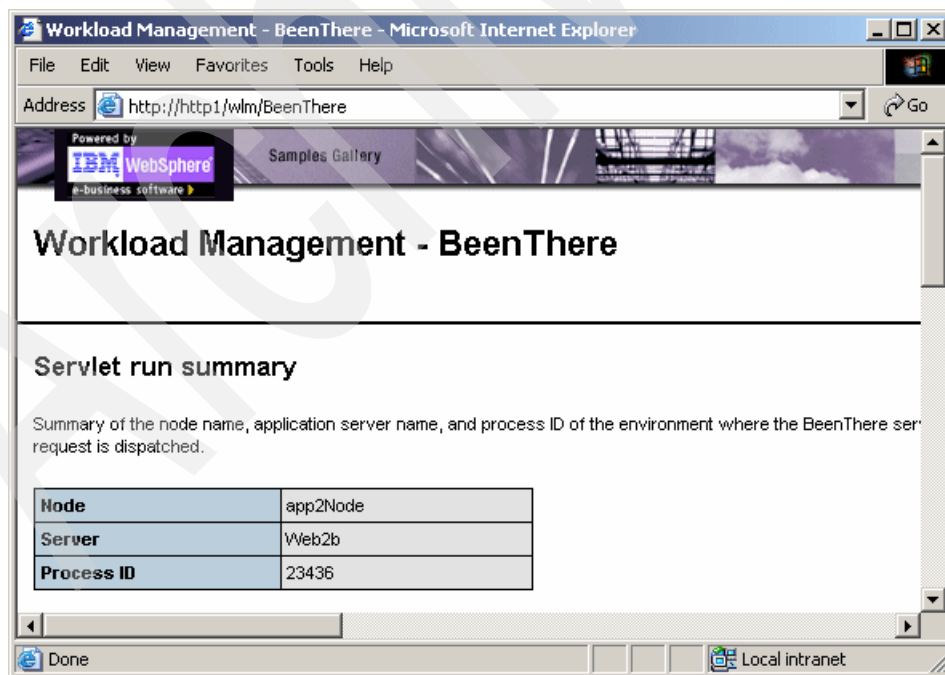
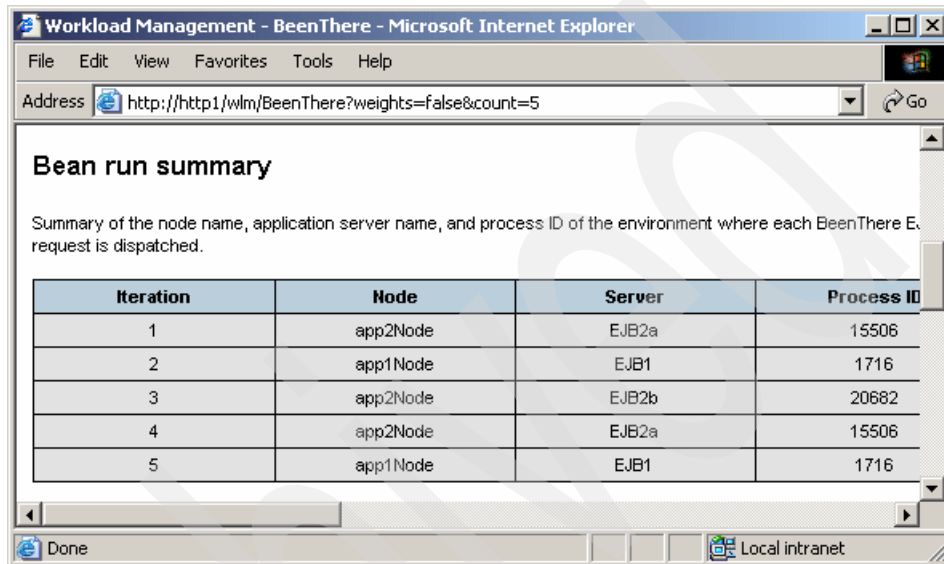


Figure 6-38 *BeenThere* first execution

- Click the **Run** button as many times as you have cluster members, and you should see from the Server name that each request (clicking **Run**) is routed to a different cluster member (since no session is created, there is no affinity). Eventually all cluster members will be cycled through. You can also ask for several bean executions at once, with results as shown in Figure 6-39.



Iteration	Node	Server	Process ID
1	app2Node	EJB2a	15506
2	app1Node	EJB1	1716
3	app2Node	EJB2b	20682
4	app2Node	EJB2a	15506
5	app1Node	EJB1	1716

Figure 6-39 *BeenThere* response for 5 executions

Note: Changing the bean iteration will not round robin the servlet request. There is only one Web request and multiple EJB requests.

- Now change the load balancing option for the Web server *http1* from round-robin to random using the Administrative Console.
- Save and generate/propagate the plugin-cfg.xml file.
- Restart the Web server or wait for the plug-in to reload the configuration file. Go back to the browser and click **Run** again.
- The requests will now be served in a random order, which could mean that all clicks of the **Run** button go to the same cluster member.

Testing session management

As described before, there are three ways of tracking session management:

- ▶ Cookies
- ▶ SSL ID
- ▶ URL rewriting

Cookies

To test session affinity using cookies do the following:

1. In the http1 Web server configuration, change the plug-in log settings. Set the plug-in log level to Trace and filename to, for example, sesstrace.log.
2. In the Administrative Console, click **Servers -> Application servers -> <AppServer_Name> -> Web Container Settings -> Session management** (in this example we choose the Web1 Application Server). The window shown in Figure 6-40 appears.
3. Check the **Enable Cookies** check box (if not already checked).
4. Save and generate/propagate the plugin-cfg.xml file.
5. Repeat steps 2 to 4 for the other application servers in the cluster. In our case, also servers Web2a and Web2b need updating. Save and synchronize the changes with all nodes.

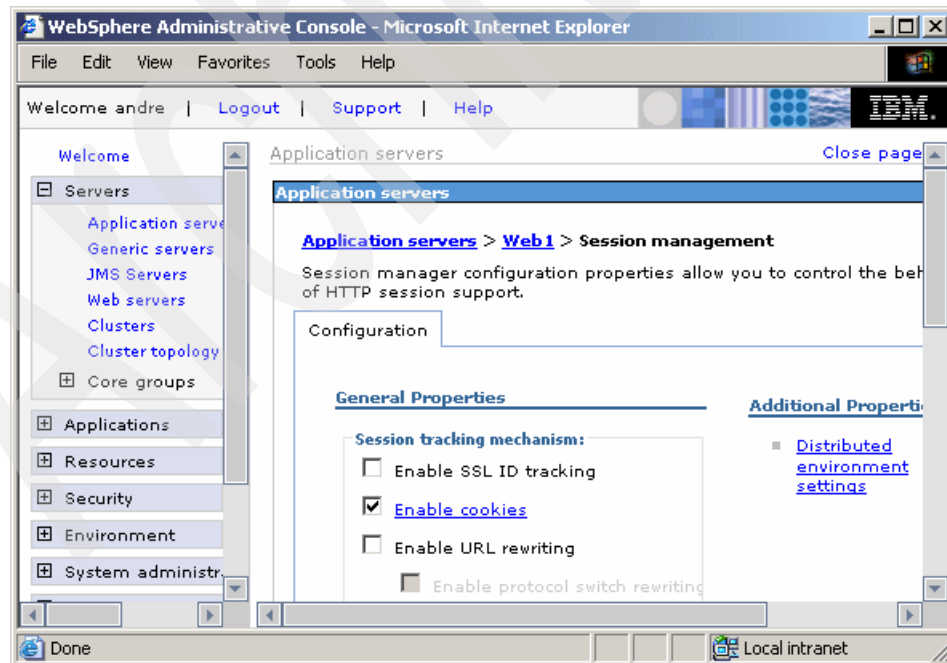


Figure 6-40 Setting Cookies for session management.

6. Stop and start the cluster to make the session manager changes in all servers operational.
7. Open a browser and go to the following URL (again, it is assumed that the DefaultApplication.ear has already been installed in the cluster):
`http://http1/hitcount`
8. Select **Session state** from the Select a method of execution options, as shown in Figure 6-41.

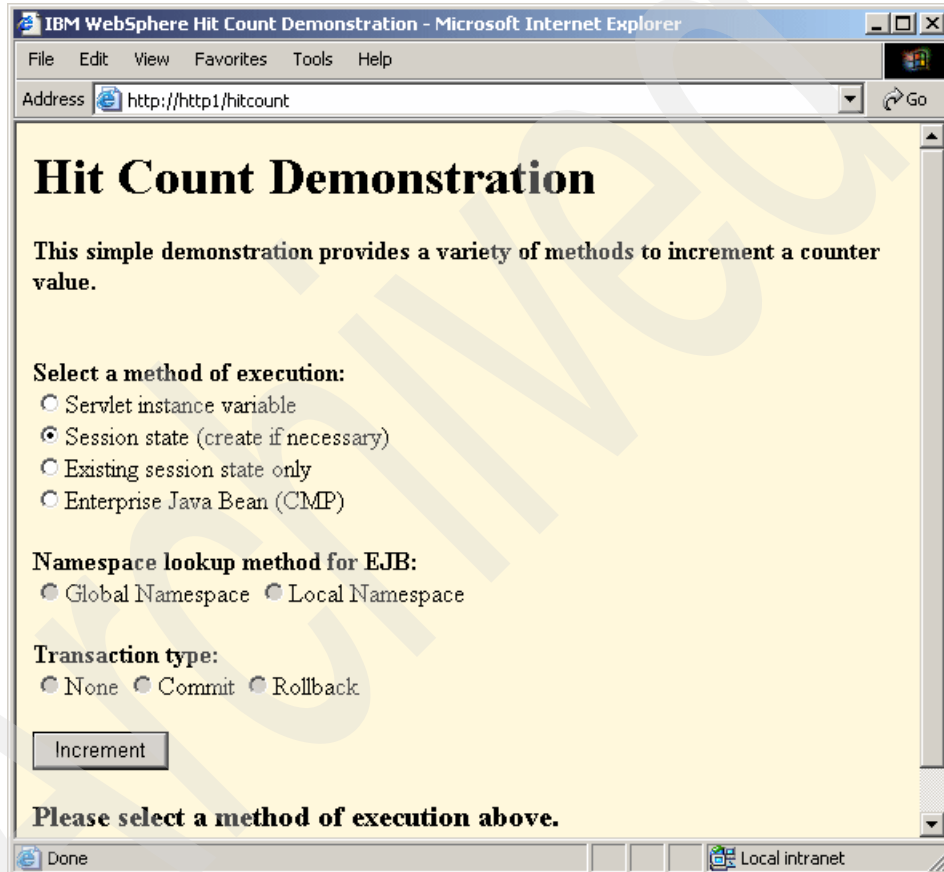


Figure 6-41 HitCount servlet

9. Click the **Increment** button a few times (always selecting the **Session state** option before).
10. Open your plug-in log as specified in step 1 on page 312. Look through the processing of the requests and search for `websphereHandleSessionAffinity`. You should see something similar to that shown in Example 6-13.

Note: If the plug-in log does not contain any trace information, the refresh interval for the plug-in may not have been reached. Wait longer or restart the Web server.

11. Change the Log tag back to its original setting.

Example 6-13 Plug-in trace when using cookies

```
...
ws_common: websphereWriteRequestReadResponse: Enter
ws_common: websphereHandleSessionAffinity: Checking for session affinity
ws_common: websphereHandleSessionAffinity: Checking the SSL session id
lib_htrequest: htrequestGetCookieValue: Looking for cookie: 'SSLJSESSION'
lib_htrequest: htrequestGetCookieValue: No cookie found for: 'SSLJSESSION'
ws_common: websphereHandleSessionAffinity: Checking the cookie affinity: JSESSIONID
lib_htrequest: htrequestGetCookieValue: Looking for cookie: 'JSESSIONID'
lib_htrequest: htrequestGetCookieValue: name='JSESSIONID',
value='0000zWhbR5Dz12Egzy9A8911iSk:vtnu19n9'
ws_common: websphereParseCloneID: Parsing clone ids from '0000zWhbR5Dz12Egzy9A8911iSk:vtnu19n9'
ws_common: websphereParseCloneID: Adding clone id 'vtnu19n9'
ws_common: websphereParseCloneID: Returning list of clone ids
ws_server_group: serverGroupFindClone: Looking for clone
ws_server_group: serverGroupGetFirstPrimaryServer: getting the first primary server
ws_server_group: serverGroupFindClone: Comparing curCloneID 'vtnu19n9' to server clone id
'vtnu14vu'
ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
ws_server_group: serverGroupFindClone: Comparing curCloneID 'vtnu19n9' to server clone id
'vtnu19n9'
ws_server_group: serverGroupFindClone: Match for clone 'app2Node_Web2a'
ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0, maxConnectionsCount -1.
ws_server_group: serverGroupCheckServerStatus: Checking status of app2Node_Web2a, ignoreWeights
1, markedDown 0, retryNow 0, wlbAllows 0 reachedMaxConnectionsLimit 0
ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0, maxConnectionsCount -1.
ws_server_group: serverGroupIncrementConnectionCount: Server app2Node_Web2a picked,
pendingConnectionCount 1 totalConnectionsCount 3.
ws_common: websphereHandleSessionAffinity: Setting server to app2Node_Web2a
ws_common: websphereFindTransport: Finding the transport
ws_common: websphereFindTransport: Setting the transport(case 2): app2.itso.ibm.com on port
9080
ws_common: websphereExecute: Executing the transaction with the app server
ws_common: websphereGetStream: Getting the stream to the app server
ws_transport: transportStreamDequeue: Checking for existing stream from the queue
ws_common: websphereSocketIsClosed: Checking to see if socket is still open
ws_common: websphereGetStream: Using existing stream from transport app2.itso.ibm.com:9080
queue, socket = 424
...
```

SSL ID

We tested session affinity using SSL ID and cookies as follows:

1. Set up your Web server for SSL traffic. See the relevant Web server documentation for details on this.
2. In the Administrative Console, go to the virtual host definitions by selecting **Environment -> Virtual Hosts -> <Virtual_Host_Name> -> Host Aliases -> New**.
3. Add the host name as http1 and the port as 443 and click **Apply**.
4. Go to **Servers -> Application servers -> <AppServer_Name> -> Web Container Settings -> Session management**. Select both **Enable SSL ID tracking** and **Enable cookies**. We did this at the application server level so that all other session management levels would be overwritten. However, if you have overwritten session management set at the application or Web module level this will not work.
5. Click **Apply**, save and synchronize the changes.
6. Repeat steps 4 and 5 for all other members of the cluster.
7. Stop and start your cluster to make the session manager changes operational.
8. In the http1 Web server configuration, change the plug-in log settings. Set the plug-in log level to Trace and filename to, for example, sesstrace.log.
9. Regenerate/propagate the plug-in configuration file for your http1 Web server.
10. Open a browser and go to:
`https://http1/hitcount`
Make sure to connect using https from your Web browser.
11. From the Select a method of execution option, select **Session state** (see Figure 6-41 on page 313).
12. Click the **Increment** button a few times.
13. Open your plug-in log as specified in step 8. Look through the processing of the requests and search for websphereHandleSessionAffinity.
You should see something similar to that shown in Example 6-14 on page 316.
14. Change the Log filename back to its original setting.

Note: When using SSL between the client and the Web server, WebSphere Application Server V6 by default will use SSL communication between the plug-in and the Web container also.

To change this, for example to improve performance, disable the WCInboundDefaultSecure transport chains for the server. Once the WCInboundDefaultSecure transport chain is disabled, the session management will send JSESSIONID instead of SSLJSESSION to the Web container.

Example 6-14 Plug-in trace when using SSL ID and cookies

```
...
TRACE: ws_common: websphereWriteRequestReadResponse: Enter
TRACE: ws_common: websphereHandleSessionAffinity: Checking for session affinity
TRACE: ws_common: websphereHandleSessionAffinity: Checking the SSL session id
TRACE: lib_htrequest: htrequestGetCookieValue: Looking for cookie: 'SSLJSESSION'
TRACE: lib_htrequest: htrequestGetCookieValue: name='SSLJSESSION',
value='0001SESSIONMANAGEMENTAFFINI:103u93ko1'
TRACE: ws_common: websphereParseCloneID: Parsing clone ids from
'0001SESSIONMANAGEMENTAFFINI:103u93ko1'
TRACE: ws_common: websphereParseCloneID: Adding clone id '103u93ko1'
TRACE: ws_common: websphereParseCloneID: Returning list of clone ids
TRACE: ws_server_group: serverGroupFindClone: Looking for clone
TRACE: ws_server_group: serverGroupGetFirstPrimaryServer: getting the first primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID '103u93ko1' to server clone
id '103slf91d'
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID '103u93ko1' to server clone
id '103slfaf2'
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID '103u93ko1' to server clone
id '103u93ko1'
TRACE: ws_server_group: serverGroupFindClone: Match for clone 'app1Node_Web1'
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0,
maxConnectionsCount -1.
STATS: ws_server_group: serverGroupCheckServerStatus: Checking status of app1Node_Web1,
ignoreWeights 1, markedDown 0, retryNow 0, wlbAllows 0 reachedMaxConnectionsLimit 0
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0,
maxConnectionsCount -1.
TRACE: ws_server_group: serverGroupIncrementConnectionCount: Server app1Node_Web1 picked,
pendingConnectionCount 1 totalConnectionsCount 49.
TRACE: ws_common: websphereHandleSessionAffinity: Setting server to app1Node_Web1
...
TRACE: ws_server_group: lockedServerGroupUseServer: Server app1Node_Web1 picked, weight 0.
TRACE: ws_common: websphereFindTransport: Finding the transport
```

```
TRACE: ws_common: websphereFindTransport: Setting the transport(case 1): appl.itso.ibm.com on
port 9447
TRACE: ws_common: websphereExecute: Executing the transaction with the app server
TRACE: ws_common: websphereGetStream: Getting the stream to the app server
TRACE: ws_transport: transportStreamDequeue: Checking for existing stream from the queue
TRACE: ws_common: websphereSocketIsClosed: Checking to see if socket is still open
TRACE: ws_common: websphereGetStream: Using existing stream from transport
appl.itso.ibm.com:9447 queue, socket = 16904
```

URL rewriting

There is no servlet provided with the WebSphere samples that uses URL rewriting. To perform this test, you can either use your own example or the sample given in Appendix A, “Sample URL rewrite servlet” on page 1033.

We tested session affinity using the URL rewrite sample from the Appendix as follows:

1. Set up the URL rewrite sample application. Make sure you regenerate the plug-in configuration on your Web server.
2. In the http1 Web server configuration, change the plug-in log settings. Set the plug-in log level to Trace and filename to sesstrace.log.
3. Go to **Servers -> Application servers -> <AppServer_Name> -> Web Container Settings -> Session Management**. In this example, the Web1 Application Server is first selected, as shown in Figure 6-42 on page 318.
4. Click the **Enable URL Rewriting** check box.

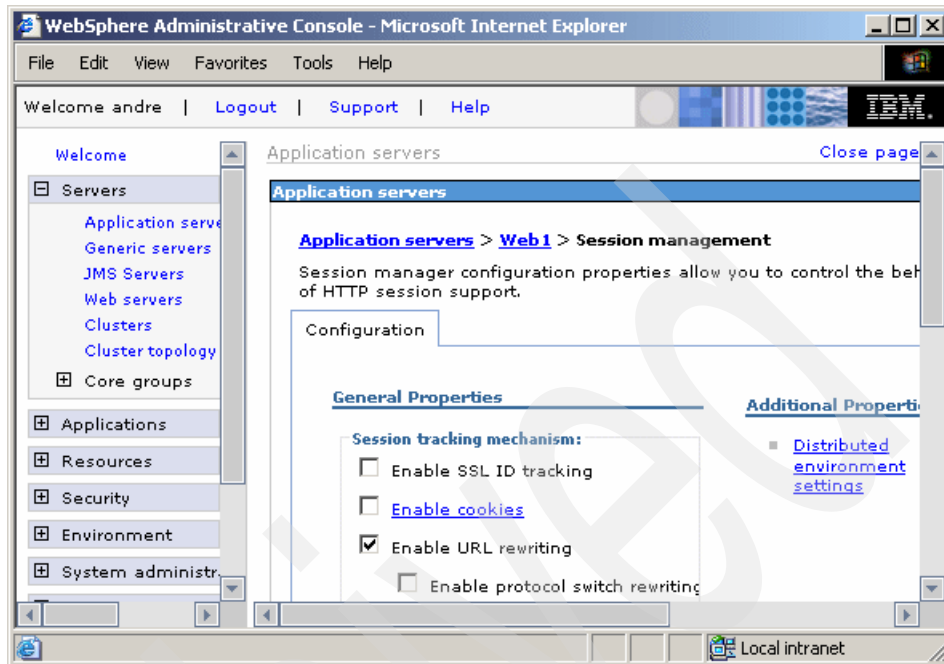


Figure 6-42 Setting URL rewriting for session management.

5. Repeat steps 3 on page 317 and 4 on page 317 for the rest of the cluster members.
6. Save and synchronize the nodes.
7. Regenerate and propagate the http1 Web server plug-in configuration file.
8. Stop and start your cluster to make the session manager changes operational.
9. Open a browser and go to:
 http://http1/ur1test/ur1test
10. Click the **Request this servlet again using the rewritten URL** link, as shown in Figure 6-43 on page 319.
11. Click the **Request this servlet again using the rewritten URL** link a few more times.

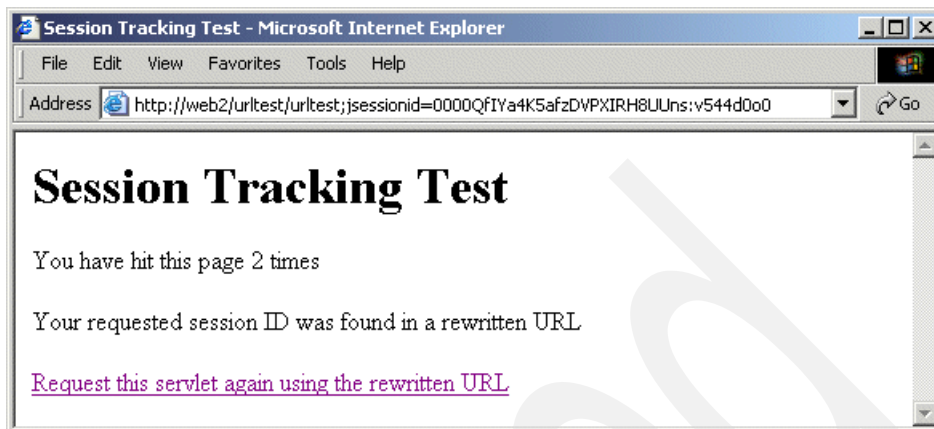


Figure 6-43 URL rewriting example

12. Open your plug-in log as specified in step 2 on page 317. Look through the processing of the requests and search for `websphereHandleSessionAffinity`. You should see something similar to that shown in Example 6-15.
13. Change the Log tag back to its original setting.

Example 6-15 Plug-in trace when using URL rewriting

```
...
ws_common: websphereWriteRequestReadResponse: Enter
ws_common: websphereHandleSessionAffinity: Checking for session affinity
ws_common: websphereHandleSessionAffinity: Checking the SSL session id
lib_htrequest: htrequestGetCookieValue: Looking for cookie: 'SSLJSESSION'
lib_htrequest: htrequestGetCookieValue: No cookie found for: 'SSLJSESSION'
ws_common: websphereHandleSessionAffinity: Checking the cookie affinity: JSESSIONID
lib_htrequest: htrequestGetCookieValue: Looking for cookie: 'JSESSIONID'
lib_htrequest: htrequestGetCookieValue: No cookie found for: 'JSESSIONID'
ws_common: websphereHandleSessionAffinity: Checking the url rewrite affinity: jsessionid
ws_common: websphereParseSessionID: Parsing session id from
'/urltest/urltest;jsessionid=0000evXHbbzKzoQlDGhb7U0abn:103u93ko1'
ws_common: websphereParseSessionID: Parsed session id
'jsessionid=0000evXHbbzKzoQlDGhb7U0abn:103u93ko1'
ws_common: websphereParseCloneID: Parsing clone ids from
'/urltest/urltest;jsessionid=0000evXHbbzKzoQlDGhb7U0abn:103u93ko1'
ws_common: websphereParseCloneID: Adding clone id '103u93ko1'
ws_common: websphereParseCloneID: Returning list of clone ids
ws_server_group: serverGroupFindClone: Looking for clone
ws_server_group: serverGroupGetFirstPrimaryServer: getting the first primary server
ws_server_group: serverGroupFindClone: Comparing curCloneID '103u93ko1' to server clone id
'103s1f91d'
ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
```

```

ws_server_group: serverGroupFindClone: Comparing curCloneID '103u93kol' to server clone id
'103slfaf2'
ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
ws_server_group: serverGroupFindClone: Comparing curCloneID '103u93kol' to server clone id
'103u93kol'
ws_server_group: serverGroupFindClone: Match for clone 'applNode_Web1'
ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0, maxConnectionsCount -1.
[Wed Dec 22 18:20:43 2004] 00000eec 00000104 - STATS: ws_server_group:
serverGroupCheckServerStatus: Checking status of applNode_Web1, ignoreWeights 1, markedDown 0,
retryNow 0, wlbAllows 0 reachedMaxConnectionsLimit 0
ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0, maxConnectionsCount -1.
ws_server_group: serverGroupIncrementConnectionCount: Server applNode_Web1 picked,
pendingConnectionCount 1 totalConnectionsCount 2.
ws_common: websphereHandleSessionAffinity: Setting server to applNode_Web1
ws_server_group: assureWeightsValid: group WEBcluster
ws_server_group: serverGroupGetFirstPrimaryServer: getting the first primary server
ws_server_group: weights_need_reset: app2Node_Web2a: 1 max, 1 cur.
ws_server_group: lockedServerGroupUseServer: Server applNode_Web1 picked, weight -1.
ws_common: websphereFindTransport: Finding the transport
ws_common: websphereFindTransport: Setting the transport(case 2): appl.itso.ibm.com on port
9082
ws_common: websphereExecute: Executing the transaction with the app server
ws_common: websphereGetStream: Getting the stream to the app server
ws_transport: transportStreamDequeue: Checking for existing stream from the queue
ws_common: websphereSocketIsClosed: Checking to see if socket is still open
ws_common: websphereGetStream: Using existing stream from transport appl.itso.ibm.com:9082
queue, socket = 10712

```

6.10.2 Failover operation

This section shows what happens when certain failover situations are forced with the Web container. These tests were performed in our lab, using the sample topology described in Chapter 8, “Implementing the sample topology” on page 387.

We stopped the second Web server, http2, to force all browser requests to http1. This way, we only need to observe the plug-in behavior on http1.

The following failure scenarios are covered:

- ▶ Stopping a cluster member
- ▶ Stopping a cluster member containing active sessions
- ▶ Killing a cluster member during a request
- ▶ Overloading a cluster member

Stopping a cluster member

We tested plug-in failover operation with a stopped cluster member, as follows:

1. Verify that all the cluster members are running within the cluster. Check this by following the steps described in “Workload management with the plug-in” on page 310.

Cycle through to make sure that all of the cluster members are available. There is no need to repeat the changes to the workload management policy.
2. In the http1 Web server configuration, change the plug-in log settings. Set the plug-in log level to Trace and filename to, for example, `stopclustermember.log`.
3. Save and synchronize the nodes.
4. Regenerate and propagate the http1 Web server plug-in configuration file.
5. Open the Administrative Console and select one of your cluster members.
6. Stop this cluster member.
7. Repeat step 1, noting the absence of the cluster member just stopped.
8. Start the cluster member again.
9. Wait 60 seconds and repeat step 1. The cluster member should return to serving requests.

What is happening?

The plug-in uses the round robin (or random) method to distribute the requests to the cluster members. Upon reaching the cluster member that was stopped, the plug-in attempts to connect and finds there is no HTTP process listening on the port.

The plug-in marks this cluster member as down and writes an error to the log, as shown in Example 6-16.

Example 6-16 Plug-in trace with cluster member down

```
...
ERROR: ws_common: websphereGetStream: Failed to connect to app server on host
'app2.itso.ibm.com', OS err=10061
TRACE: ws_common: websphereGetStream: socket 10712 closed - failed to connect
ERROR: ws_common: websphereExecute: Failed to create the stream
ERROR: ws_server: serverSetFailoverStatus: Marking app2Node_Web2b down
STATS: ws_server: serverSetFailoverStatus: Server app2Node_Web2b :
pendingConnections 0 failedConnections 1 affinityConnections 0 totalConnections
0.
```

```
ERROR: ws_common: websphereHandleRequest: Failed to execute the transaction to
'app2Node_Web2b' on host 'app2.itso.ibm.com'; will try another one
...
```

It then tries to connect to the next cluster member in the primary server list. When it has found a cluster member that works, the request is served from that cluster member instead.

The plug-in does not try the cluster member for another 60 seconds. If tracing is enabled, you will be able to see that the plug-in shows the time left every time it comes to the downed cluster member in the round robin algorithm, as shown in Example 6-17.

Example 6-17 Plug-in trace cluster member retry interval countdown

```
...
STATS: ws_server_group: serverGroupCheckServerStatus: Checking status of
app2Node_Web2b, ignoreWeights 0, markedDown 1, retryNow 0, wlbAllows 0
reachedMaxConnectionsLimit 0
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0,
maxConnectionsCount -1.
TRACE: ws_server_group: serverGroupCheckServerStatus: Server app2Node_Web2b is
marked down; retry in 23
...
```

After restarting the cluster member and once the 60-second retry interval has passed, the next request attempt to the downed cluster member tries to connect again. This time, it is successful and the request is served.

Stopping a cluster member containing active sessions

When using sessions, the impact of stopping or killing a cluster member is that session information in that cluster member is lost. This means that persistent or replicated sessions need to be used to allow for the failover of the session to another cluster member. Follow this procedure to discover what happens:

1. Make sure persistent or replicated sessions have been enabled. See 6.8.3, “Session management and failover inside the plug-in” on page 285 for details.
2. Set up and test session tracking using cookies, as described in “Cookies” on page 312.
3. Verify that all the cluster members are running within the cluster. Check this by following the steps described in “Workload management with the plug-in” on page 310.

Cycle through to make sure that all the cluster members are available. There is no need to repeat the changes to the workload management policy.

Note: If you use the same browser for the session management test and the workload management test, you will find that all your BeenThere requests return to the same server. This is because the cookie for your session management test is still valid.

4. Return to the session tracking test at:

`http://http1/hitcount`

Click the **Increment** button a few times and remember the session count you have reached.

5. Check the log to see which cluster member served your session, referring to Example 6-13 on page 314 to see what to look for. Alternatively:

- a. From the same browser, point to:

`http://http1/wlm/BeenThere`

- b. Note the servlet server name, since this is the cluster member that is serving your session.

- c. Return to the session tracking test at:

`http://http1/hitcount`

6. Stop the cluster member that you located in the log file or using BeenThere.
7. Increment the session count in your Web browser. You will see that the session count should continue from the number you noted in step 4.
8. Start the downed cluster member again.
9. Repeat step 7. The session counter will still continue from the previous number and no new session will be created.

What is happening?

Upon choosing to increment the session counter, WebSphere adds a cookie to your browser. Within this cookie is the session identifier and a clone ID (or partition ID) of where this session is stored. This is the expected behavior that makes session affinity possible, as previously discussed.

At the end of each request (or at the specified interval as described in 6.8.8, “Session management tuning” on page 300), the session is written to the database or distributed between the configured number of application servers using memory-to-memory replication. When the cluster member running the session is stopped, the plug-in chooses another cluster member for the request to go to instead. In the case of using database persistence, this cluster member, finding that it does not have the session cached locally, searches the database.

In the case of using memory-to-memory replication, the application server will find the session in its own cache. So in both cases, the failover application server finds the session and uses that one instead of creating a new one. This is why the session counter is incremented, not reset.

You can verify this behavior when looking at the following plug-in traces. Example 6-18 shows the plug-in activity when distributed sessions rely on database persistence (or when using memory-to-memory replication in any other configuration than peer-to-peer). When an incoming request finds its server down, the WebSphere session manager changes the session cookie and appends the cluster clone ID of the new cluster member to it. The cookie now has two cluster clone IDs on it, the original (stopped) cluster member and the failover cluster member.

The plug-in now tries to connect to the stopped cluster member first, and finding it marked as down, will try the failover cluster member instead.

Starting up the stopped cluster member means that the plug-in now returns the session requests to the original cluster member after the retry interval has passed.

Example 6-18 Plug-in trace with session failover (database persistence)

```
...
### We have a session with clone id '103u93ko1' for app1Node_Web1.
TRACE: ws_server: serverCreate: Creating the server object
TRACE: ws_server: serverSetName: Setting name app1Node_Web1
TRACE: ws_server: serverSetCloneID: Setting clone id 103u93ko1
...
### We have a session with clone id '103s1f91d' for app2Node_Web2a.
TRACE: ws_server: serverCreate: Creating the server object
TRACE: ws_server: serverSetName: Setting name app2Node_Web2a
TRACE: ws_server: serverSetCloneID: Setting clone id 103s1f91d
...
### When a request with session affinity comes, the plugin dispatches it to Web1.
TRACE: ws_common: websphereHandleSessionAffinity: Checking the cookie affinity: JSESSIONID
TRACE: lib_htrequest: htrequestGetCookieValue: Looking for cookie: 'JSESSIONID'
TRACE: lib_htrequest: htrequestGetCookieValue: name='JSESSIONID',
value='000109Ra5SyRIRoycNXAnKusKEb:103u93ko1'
TRACE: ws_common: websphereParseCloneID: Parsing clone ids from
'000109Ra5SyRIRoycNXAnKusKEb:103u93ko1'
TRACE: ws_common: websphereParseCloneID: Adding clone id '103u93ko1'
TRACE: ws_common: websphereParseCloneID: Returning list of clone ids
TRACE: ws_server_group: serverGroupFindClone: Looking for clone
TRACE: ws_server_group: serverGroupGetFirstPrimaryServer: getting the first primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID '103u93ko1' to server clone
id '103s1f91d'
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
```

```

TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID '103u93ko1' to server clone
id '103slfaf2'
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID '103u93ko1' to server clone
id '103u93ko1'
TRACE: ws_server_group: serverGroupFindClone: Match for clone 'applNode_Web1'
...
### When the Web1 server is stopped it is marked as down.
TRACE: ws_transport: transportStreamDequeue: Checking for existing stream from the queue
TRACE: ws_common: websphereSocketIsClosed: Checking to see if socket is still open
TRACE: ws_common: websphereSocketIsClosed: socket 460 was closed by peer
TRACE: ws_common: websphereGetStream: Destroying queued stream; socket already closed
TRACE: lib_stream: destroyStream: Destroying the stream
TRACE: lib_rio: rclose: socket 460 closed
TRACE: ws_transport: transportStreamDequeue: Checking for existing stream from the queue
ERROR: ws_common: websphereGetStream: Failed to connect to app server on host
'appl.itso.ibm.com', OS err=10061
TRACE: ws_common: websphereGetStream: socket 460 closed - failed to connect
ERROR: ws_common: websphereExecute: Failed to create the stream
ERROR: ws_server: serverSetFailoverStatus: Marking applNode_Web1 down
STATS: ws_server: serverSetFailoverStatus: Server applNode_Web1 : pendingConnections 0
failedConnections 2 affinityConnections 36 totalConnections 37.
ERROR: ws_common: websphereHandleRequest: Failed to execute the transaction to
'applNode_Web1' on host 'appl.itso.ibm.com'; will try another one
...
### The plugin iterates again descending through the primary server list until it finds an "up"
### server (Web2a), who receives the request from the plug-in.
TRACE: lib_htrequest: htrequestGetCookieValue: Looking for cookie: 'JSESSIONID'
TRACE: lib_htrequest: htrequestGetCookieValue: name='JSESSIONID',
value='000109Ra5SyRIRoycNXAnKusKEb:103u93ko1'
TRACE: ws_common: websphereParseCloneID: Parsing clone ids from
'000109Ra5SyRIRoycNXAnKusKEb:103u93ko1'
TRACE: ws_common: websphereParseCloneID: Adding clone id '103u93ko1'
TRACE: ws_common: websphereParseCloneID: Returning list of clone ids
TRACE: ws_server_group: serverGroupFindClone: Looking for clone
TRACE: ws_server_group: serverGroupGetFirstPrimaryServer: getting the first primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID '103u93ko1' to server clone
id '103slf9ld'
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID '103u93ko1' to server clone
id '103slfaf2'
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID '103u93ko1' to server clone
id '103u93ko1'
TRACE: ws_server_group: serverGroupFindClone: Match for clone 'applNode_Web1'
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0,
maxConnectionsCount -1.
STATS: ws_server_group: serverGroupCheckServerStatus: Checking status of applNode_Web1,
ignoreWeights 1, markedDown 1, retryNow 0, wlbAllows 0 reachedMaxConnectionsLimit 0

```

```

TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0,
maxConnectionsCount -1.
TRACE: ws_server_group: serverGroupCheckServerStatus: Server app1Node_Web1 is marked down;
retry in 60
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
TRACE: ws_server_group: serverGroupGetNextUpPrimaryServer: Getting the next up primary server
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
TRACE: ws_server_group: serverGroupGetFirstPrimaryServer: getting the first primary server
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0,
maxConnectionsCount -1.
STATS: ws_server_group: serverGroupCheckServerStatus: Checking status of app2Node_Web2a,
ignoreWeights 1, markedDown 0, retryNow 0, wlbAllows 11 reachedMaxConnectionsLimit 0
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0,
maxConnectionsCount -1.
TRACE: ws_server_group: serverGroupIncrementConnectionCount: Server app2Node_Web2a picked,
pendingConnectionCount 1 totalConnectionsCount 26.
TRACE: ws_server_group: serverGroupFindClone: Returning next up server app2Node_Web2a
TRACE: ws_common: websphereHandleSessionAffinity: Setting server to app2Node_Web2a
...
### On the response back from Web2a it's cloneID is appended to the session cookie.
TRACE: lib_htrequest: htrequestWrite: Writing the request:
TRACE: GET /urltest/urltest HTTP/1.1
TRACE: User-Agent: Opera/6.05 (Windows XP; U) [en]
TRACE: Host: http1
...
TRACE: lib_htresponse: htresponseSetContentLength: Setting the content length |301|
TRACE: Set-Cookie: JSESSIONID=000209Ra5SyRIRoycNXAnKusKEb:103u93ko1:103s1f91d; path=/
...
### On the next incoming request from this client we have a second clone id on the session
### cookie. Web1 will be skipped until the retry interval has elapsed.
TRACE: ws_common: websphereHandleSessionAffinity: Checking the cookie affinity: JSESSIONID
TRACE: lib_htrequest: htrequestGetCookieValue: Looking for cookie: 'JSESSIONID'
TRACE: lib_htrequest: htrequestGetCookieValue: name='JSESSIONID',
value='000207eBLC1K7y5v0tGuRcASBVd:103u93ko1:103s1f91d'
TRACE: ws_common: websphereParseCloneID: Parsing clone ids from
'000207eBLC1K7y5v0tGuRcASBVd:103u93ko1:103s1f91d'
TRACE: ws_common: websphereParseCloneID: Adding clone id '103u93ko1'
TRACE: ws_common: websphereParseCloneID: Adding clone id '103s1f91d'
TRACE: ws_common: websphereParseCloneID: Returning list of clone ids
TRACE: ws_server_group: serverGroupFindClone: Looking for clone
TRACE: ws_server_group: serverGroupGetFirstPrimaryServer: getting the first primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID '103u93ko1' to server clone
id '103s1f91d'
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID '103u93ko1' to server clone
id '103s1faf2'
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID '103u93ko1' to server clone
id '103u93ko1'

```



```

TRACE: ws_server_group: serverGroupFindClone: Match for clone 'app1Node_Web1'
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0,
maxConnectionsCount -1.
STATS: ws_server_group: serverGroupCheckServerStatus: Checking status of app1Node_Web1,
ignoreWeights 1, markedDown 1, retryNow 0, wlbAllows 0 reachedMaxConnectionsLimit 0
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0,
maxConnectionsCount -1.
TRACE: ws_server_group: serverGroupCheckServerStatus: Server app1Node_Web1 is marked down;
retry in 50
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
TRACE: ws_server_group: serverGroupGetFirstPrimaryServer: getting the first primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID '103s1f91d' to server clone
id '103s1f91d'
TRACE: ws_server_group: serverGroupFindClone: Match for clone 'app2Node_Web2a'
...
### When Web1 comes back to life after the retry interval our session requests go back to it
TRACE: ws_common: websphereHandleSessionAffinity: Checking the cookie affinity: JSESSIONID
TRACE: lib_htrequest: htrequestGetCookieValue: Looking for cookie: 'JSESSIONID'
TRACE: lib_htrequest: htrequestGetCookieValue: name='JSESSIONID',
value='000207eBLC1K7y5v0tGuRcASBVd:103u93ko1:103s1f91d'
TRACE: ws_common: websphereParseCloneID: Parsing clone ids from
'000207eBLC1K7y5v0tGuRcASBVd:103u93ko1:103s1f91d'
TRACE: ws_common: websphereParseCloneID: Adding clone id '103u93ko1'
TRACE: ws_common: websphereParseCloneID: Adding clone id '103s1f91d'
TRACE: ws_common: websphereParseCloneID: Returning list of clone ids
TRACE: ws_server_group: serverGroupFindClone: Looking for clone
TRACE: ws_server_group: serverGroupGetFirstPrimaryServer: getting the first primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID '103u93ko1' to server clone
id '103s1f91d'
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID '103u93ko1' to server clone
id '103s1faf2'
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID '103u93ko1' to server clone
id '103u93ko1'
TRACE: ws_server_group: serverGroupFindClone: Match for clone 'app1Node_Web1'
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0,
maxConnectionsCount -1.
STATS: ws_server_group: serverGroupCheckServerStatus: Checking status of app1Node_Web1,
ignoreWeights 1, markedDown 1, retryNow 1, wlbAllows 0 reachedMaxConnectionsLimit 0
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0,
maxConnectionsCount -1.
TRACE: ws_server_group: serverGroupCheckServerStatus: Time to retry server app1Node_Web1
TRACE: ws_server_group: serverGroupIncrementConnectionCount: Server app1Node_Web1 picked,
pendingConnectionCount 1 totalConnectionsCount 43.
TRACE: ws_common: websphereHandleSessionAffinity: Setting server to app1Node_Web1
TRACE: ws_server_group: assureWeightsValid: group WEBcluster
TRACE: ws_server_group: serverGroupGetFirstPrimaryServer: getting the first primary server
TRACE: ws_server_group: weights_need_reset: app2Node_Web2a: 1 max, -7 cur.

```

```
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
TRACE: ws_server_group: weights_need_reset: app2Node_Web2b: 1 max, 2 cur.
TRACE: ws_server_group: lockedServerGroupUseServer: Server app1Node_Web1 picked, weight -1.
TRACE: ws_common: websphereFindTransport: Finding the transport
```

Looking at another trace, shown in Example 6-19, you can see the plug-in activity when distributed sessions rely on memory-to-memory replication in the peer-to-peer mode. Instead of a clone ID, the session cookie holds a partition ID that maps to the partition table known by the plug-in.

When the plug-in finds a server down, it acquires a new partition table from another application server and then resends the request to a cluster member that holds a session replica. For more information about partition ID, please see “Partition ID” on page 297.

Example 6-19 Plug-in trace with session failover (memory-to-memory with peer-to-peer configuration)

```
...
### We have a session with clone id '103s1f91d' for app2Node_Web2a.
TRACE: ws_server: serverCreate: Creating the server object
TRACE: ws_server: serverSetName: Setting name app2Node_Web2a
TRACE: ws_server: serverSetCloneID: Setting clone id 103s1f91d
...
### We have a session with clone id '103u93ko1' for app1Node_Web1.
TRACE: ws_server: serverCreate: Creating the server object
TRACE: ws_server: serverSetName: Setting name app1Node_Web1
TRACE: ws_server: serverSetCloneID: Setting clone id 103u93ko1
...
### When a request with session affinity comes, the plugin dispatches it to Web1.
TRACE: ws_common: websphereHandleSessionAffinity: Checking the cookie affinity: JSESSIONID
TRACE: lib_htrequest: htrequestGetCookieValue: Looking for cookie: 'JSESSIONID'
TRACE: lib_htrequest: htrequestGetCookieValue: name='JSESSIONID',
value='00018fMFv4U4msm2Gq12tW2JMOI:-1101060518'
TRACE: ws_common: websphereParseCloneID: Parsing clone ids from
'00018fMFv4U4msm2Gq12tW2JMOI:-1101060518'
TRACE: ws_common: websphereParseCloneID: Adding clone id '-1101060518'
TRACE: ws_common: websphereParseCloneID: Returning list of clone ids
TRACE: ws_server_group: serverGroupFindClone: Looking for clone
TRACE: ws_server_group: serverGroupGetFirstPrimaryServer: getting the first primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID '-1101060518' to server
clone id '103s1f91d'
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID '-1101060518' to server
clone id '103s1faf2'
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID '-1101060518' to server
clone id '103u93ko1'
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
```

```

TRACE: ws_server_group: serverGroupFindClone: Failed to find server that matched the clone id
TRACE: ws_common: websphereHandleSessionAffinity: Checking the url rewrite affinity: jsessionid
TRACE: ws_common: websphereParseSessionID: Parsing session id from '/urltest/urltest'
TRACE: ws_common: websphereParseSessionID: Failed to parse session id
TRACE: ws_common: websphereHandleSessionAffinity: Checking for partitionID cookie affinity: JSESSIONID
TRACE: lib_htrequest: htrequestGetCookieValue: Looking for cookie: 'JSESSIONID'
TRACE: lib_htrequest: htrequestGetCookieValue: name='JSESSIONID', value='00018fMFv4U4msm2Gq12tW2JMOI:-1101060518'
TRACE: ws_common: websphereHandleSessionAffinity: Checking the partitionID
TRACE: ws_common: websphereHandleSessionAffinity: Look for partitionID in affinity cookie JSESSIONID
TRACE: ws_common: websphereParsePartitionIDs: Parsing partition clone ids from '00018fMFv4U4msm2Gq12tW2JMOI:-1101060518'
TRACE: ws_common: websphereParsePartitionIDs: Adding clone id '-1101060518'
TRACE: ws_common: websphereParsePartitionIDs: Returning list of partition clone ids
TRACE: ws_server_group: serverGroupFindDwlmServer: Looking for dwlm pair
TRACE: ws_server_group: serverGroupMatchPartitionID: Looking for partitionID
TRACE: ws_server_group: serverGroupMatchPartitionID: Comparing curCloneID '-1101060518' to partitionID '-1352159645'
TRACE: ws_server_group: serverGroupMatchPartitionID: Comparing curCloneID '-1101060518' to partitionID '-1101060518'
TRACE: ws_server_group: serverGroupMatchPartitionID: Match found for partitionID '-1101060518'
TRACE: ws_server_group: serverGroupGetFirstServer: getting the first server
TRACE: ws_server_group: serverGroupGetServerByID: Comparing curCloneID '103u93kol' to server clone id '103u93kol'
TRACE: ws_server_group: serverGroupGetServerByID: Match for clone 'app1Node_Web1'
TRACE: ws_server_group: serverGroupFindDwlmServer: Match for clone 'app1Node_Web1'
...
### When the Web1 server is stopped it is marked as down.
TRACE: ws_transport: transportStreamDequeue: Checking for existing stream from the queue
TRACE: ws_common: websphereSocketIsClosed: Checking to see if socket is still open
TRACE: ws_common: websphereSocketIsClosed: socket 520 was closed by peer
TRACE: ws_common: websphereGetStream: Destroying queued stream; socket already closed
TRACE: lib_stream: destroyStream: Destroying the stream
TRACE: lib_rio: rclose: socket 520 closed
TRACE: ws_transport: transportStreamDequeue: Checking for existing stream from the queue
ERROR: ws_common: websphereGetStream: Failed to connect to app server on host 'app1.itso.ibm.com', OS err=10061
TRACE: ws_common: websphereGetStream: socket 520 closed - failed to connect
ERROR: ws_common: websphereExecute: Failed to create the stream
ERROR: ws_server: serverSetFailoverStatus: Marking app1Node_Web1 down
STATS: ws_server: serverSetFailoverStatus: Server app1Node_Web1 : pendingConnections 0 failedConnections 2 affinityConnections 22 totalConnections 22.
ERROR: ws_common: websphereHandleRequest: Failed to execute the transaction to 'app1Node_Web1' on host 'app1.itso.ibm.com'; will try another one
...
### The plugin retrieves an updated partition table from another member of the replication ### domain.

```

```

TRACE: ws_server_group: lockedServerGroupUseServer: Server app2Node_Web2b picked, weight 0.
TRACE: ws_server_group: serverGroupIncrementConnectionCount: Server app2Node_Web2b picked,
pendingConnectionCount 1 totalConnectionsCount 1.
TRACE: ws_server_group: serverGroupFindDwlmServer: Retrieve updated dwlm partition table from
server app2Node_Web2b (dwlmStatus = 22)
TRACE: ws_common: websphereHandleSessionAffinity: Setting server to app2Node_Web2b
...
TRACE: ws_common: websphereGetDWLMTable: Sending request to get updated partition table
...
TRACE: ws_common: ParsePartitionIDs: Parsing partitionID pair from
'-1352159645:103s1faf2;-1101060518:103s1f91d;73531583571851:103s1f91d;133506901435975:103s1f91d
';
TRACE: ws_common: ParsePartitionIDs: Adding partitionID / clone pair '-1352159645' :
'103s1faf2'
TRACE: ws_common: ParsePartitionIDs: Adding partitionID / clone pair '-1101060518' :
'103s1f91d'
TRACE: ws_common: ParsePartitionIDs: Adding partitionID / clone pair '73531583571851' :
'103s1f91d'
TRACE: ws_common: ParsePartitionIDs: Adding partitionID / clone pair '133506901435975' :
'103s1f91d'
TRACE: ws_common: ParsePartitionIDs: Returning partitionID / cloneid pair list
...
TRACE: ws_common: websphereExecute: Updated DWLM table received; dwlmStatus = 23
STATS: ws_server: serverSetFailoverStatus: Server app2Node_Web2b : pendingConnections 0
failedConnections 2 affinityConnections 2 totalConnections 0.
TRACE: ws_common: websphereHandleRequest: Updated DWLM table retrieved from 'app2Node_Web2b' on
host 'app2.itso.ibm.com'
...
### Finally the plugin finds a proper match (another cluster member from the same replication
### domain).
TRACE: ws_common: websphereHandleSessionAffinity: Checking the cookie affinity: JSESSIONID
TRACE: lib_htrequest: htrequestGetCookieValue: Looking for cookie: 'JSESSIONID'
TRACE: lib_htrequest: htrequestGetCookieValue: name='JSESSIONID',
value='00018fMFv4U4msm2Gq12tW2JMOI:-1101060518'
TRACE: ws_common: websphereParseCloneID: Parsing clone ids from
'00018fMFv4U4msm2Gq12tW2JMOI:-1101060518'
TRACE: ws_common: websphereParseCloneID: Adding clone id '-1101060518'
TRACE: ws_common: websphereParseCloneID: Returning list of clone ids
TRACE: ws_server_group: serverGroupFindClone: Looking for clone
TRACE: ws_server_group: serverGroupGetFirstPrimaryServer: getting the first primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID '-1101060518' to server
clone id '103s1f91d'
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID '-1101060518' to server
clone id '103s1faf2'
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID '-1101060518' to server
clone id '103u93ko1'
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server

```

```

TRACE: ws_server_group: serverGroupFindClone: Failed to find server that matched the clone id
TRACE: ws_common: websphereHandleSessionAffinity: Checking the url rewrite affinity: jsessionid
TRACE: ws_common: websphereParseSessionID: Parsing session id from '/urltest/urltest'
TRACE: ws_common: websphereParseSessionID: Failed to parse session id
TRACE: ws_common: websphereHandleSessionAffinity: Checking for partitionID cookie affinity:
JSESSIONID
TRACE: lib_htrequest: htrequestGetCookieValue: Looking for cookie: 'JSESSIONID'
TRACE: lib_htrequest: htrequestGetCookieValue: name='JSESSIONID',
value='00018fMFv4U4msm2Gq12tW2JMOI:-1101060518'
TRACE: ws_common: websphereHandleSessionAffinity: Checking the partitionID
TRACE: ws_common: websphereHandleSessionAffinity: Look for partitionID in affinity cookie
JSESSIONID
TRACE: ws_common: websphereParsePartitionIDs: Parsing partition clone ids from
'00018fMFv4U4msm2Gq12tW2JMOI:-1101060518'
TRACE: ws_common: websphereParsePartitionIDs: Adding clone id '-1101060518'
TRACE: ws_common: websphereParsePartitionIDs: Returning list of partition clone ids
TRACE: ws_server_group: serverGroupFindDwlmServer: Looking for dwlm pair
TRACE: ws_server_group: serverGroupMatchPartitionID: Looking for partitionID
TRACE: ws_server_group: serverGroupMatchPartitionID: Comparing curCloneID '-1101060518' to
partitionID '-1352159645'
TRACE: ws_server_group: serverGroupMatchPartitionID: Comparing curCloneID '-1101060518' to
partitionID '-1101060518'
TRACE: ws_server_group: serverGroupMatchPartitionID: Match found for partitionID '-1101060518'
TRACE: ws_server_group: serverGroupGetFirstServer: getting the first server
TRACE: ws_server_group: serverGroupGetServerByID: Comparing curCloneID '103slf9ld' to server
clone id '103u93kol'
TRACE: ws_server_group: serverGroupGetNextServer: getting the next server
TRACE: ws_server_group: serverGroupGetServerByID: Comparing curCloneID '103slf9ld' to server
clone id '103slfaf2'
TRACE: ws_server_group: serverGroupGetNextServer: getting the next server
TRACE: ws_server_group: serverGroupGetServerByID: Comparing curCloneID '103slf9ld' to server
clone id '103slf9ld'
TRACE: ws_server_group: serverGroupGetServerByID: Match for clone 'app2Node_Web2a'
TRACE: ws_server_group: serverGroupFindDwlmServer: Match for clone 'app2Node_Web2a'

```

Killing a cluster member during a request

Killing the Java virtual machine (JVM) that a cluster member is running on is the best way to see what happens if a cluster member crashes *during* a request. When killing a cluster member *before* a request, the effect is the same as seen in “Stopping a cluster member” on page 321.

When you kill a cluster member during a request, sometimes that request is dispatched to another cluster member. There is a mechanism for the plug-in to swap over the request in the middle of the process. However, this mechanism does not guarantee high-availability of all requests. In such a case, a cluster member that is functioning will be chosen for a re-issued request, which can then be tried again. The request will then be completed.

If you wish to see this, follow these steps.

1. Verify that all the cluster members are running within the cluster. Check this by following the steps described in “Workload management with the plug-in” on page 310.

Cycle through to make sure that all the cluster members are available. There is no need to repeat the changes to the workload management policy.

2. In the http1 Web server configuration, change the plug-in log settings. Set the plug-in log level to Trace and filename to, for example, `killclustermember.log`. Also make sure that the load balancing is set to use the round-robin option (For this example, we need to be able to predict the next Application Server to be picked by the plug-in on the following request, so it could also be a good idea to set equal weights on the servers just to make the test a bit easier).
3. Save and synchronize the nodes.
4. Regenerate and propagate the http1 Web server plug-in configuration file.
5. Find the Java process ID of the application server cluster member you want to kill. On AIX, we used the following command to find the process ID of our cluster member named Web2a:

```
ps -ef | grep Web2a
```

Note the process ID of your cluster member.

6. Open a browser and go to:

```
http://http1/wlm/BeenThere
```

7. Click the **Run** button until the Server name field shows the name of the cluster member *before* the one you are going to kill. Monitor how the round robin process is cycling to find this out. The next time that you click **Run**, the request needs to go to the cluster member that you intend to kill. This assumes that you are using application servers with equal weights or you need to make sure that none of your application servers reaches a weight of 0 while performing this test.
8. Set the Bean Iterations field to a large amount, for example, 10000.
9. If you are running on AIX, now is a good time to use the `tail` command to monitor your plug-in log file.
10. Click the **Run** button. This starts the request running on your cluster member. If you look at the plug-in log, you see that there is information passing between browser and application server.

11. Kill the cluster member you located in step 5 on page 332. The request will subsequently be rerouted to another cluster member. In this case in Example 6-20, the request is dispatched to another cluster member.

Example 6-20 Plug-in trace with request failover

```
...
### At first, the request is beeing processed by Web2a.
[Sun Jan 30 23:02:51 2005] 000001e4 00000ec4 - TRACE: ws_server:
serverHasReachedMaxConnections: currentConnectionsCount 0, maxConnectionsCount
-1.
[Sun Jan 30 23:02:51 2005] 000001e4 00000ec4 - TRACE: ws_server_group:
lockedServerGroupUseServer: Server app2Node_Web2a picked, weight 0.
[Sun Jan 30 23:02:51 2005] 000001e4 00000ec4 - TRACE: ws_server_group:
serverGroupIncrementConnectionCount: Server app2Node_Web2a picked,
pendingConnectionCount 1 totalConnectionsCount 65.
[Sun Jan 30 23:02:51 2005] 000001e4 00000ec4 - TRACE: ws_common:
websphereFindTransport: Finding the transport
...
### The plug-in detects the failure of Web2a and dispatches the request to
### Web2b.
[Sun Jan 30 23:02:55 2005] 000001e4 00000ec4 - TRACE: ws_common:
websphereExecute: Failed to read from an old stream; probably Keep-Alive
timeout fired
[Sun Jan 30 23:02:55 2005] 000001e4 00000ec4 - TRACE: lib_stream:
destroyStream: Destroying the stream
[Sun Jan 30 23:02:55 2005] 000001e4 00000ec4 - TRACE: lib_rio: rclose: socket
416 closed
[Sun Jan 30 23:02:55 2005] 000001e4 00000ec4 - TRACE: ws_common:
websphereGetStream: Getting the stream to the app server
[Sun Jan 30 23:02:55 2005] 000001e4 00000ec4 - TRACE: ws_transport:
transportStreamDequeue: Checking for existing stream from the queue
[Sun Jan 30 23:02:56 2005] 000001e4 00000ec4 - ERROR: ws_common:
websphereGetStream: Failed to connect to app server on host
'app2.itso.ibm.com', OS err=10061
[Sun Jan 30 23:02:56 2005] 000001e4 00000ec4 - TRACE: ws_common:
websphereGetStream: socket 416 closed - failed to connect
[Sun Jan 30 23:02:56 2005] 000001e4 00000ec4 - ERROR: ws_common:
websphereExecute: Failed to create the stream
[Sun Jan 30 23:02:56 2005] 000001e4 00000ec4 - ERROR: ws_server:
serverSetFailoverStatus: Marking app2Node_Web2a down
[Sun Jan 30 23:02:56 2005] 000001e4 00000ec4 - STATS: ws_server:
serverSetFailoverStatus: Server app2Node_Web2a : pendingConnections 0
failedConnections 5 affinityConnections 33 totalConnections 64.
[Sun Jan 30 23:02:56 2005] 000001e4 00000ec4 - ERROR: ws_common:
websphereHandleRequest: Failed to execute the transaction to 'app2Node_Web2a' on
host 'app2.itso.ibm.com'; will try another one
[Sun Jan 30 23:02:56 2005] 000001e4 00000ec4 - TRACE: ws_common:
websphereHandleSessionAffinity: Checking for session affinity
```

```

...
[Sun Jan 30 23:02:56 2005] 000001e4 00000ec4 - TRACE: ws_server_group:
serverGroupGetNextServer: getting the next server
[Sun Jan 30 23:02:56 2005] 000001e4 00000ec4 - TRACE: ws_server_group:
serverGroupGetNextPrimaryServer: getting the next primary server
[Sun Jan 30 23:02:56 2005] 000001e4 00000ec4 - TRACE: ws_server:
serverHasReachedMaxConnections: currentConnectionsCount 0, maxConnectionsCount
-1.
[Sun Jan 30 23:02:56 2005] 000001e4 00000ec4 - STATS: ws_server_group:
serverGroupCheckServerStatus: Checking status of app2Node_Web2b, ignoreWeights
0, markedDown 0, retryNow 0, wlbAllows 1 reachedMaxConnectionsLimit 0
[Sun Jan 30 23:02:56 2005] 000001e4 00000ec4 - TRACE: ws_server:
serverHasReachedMaxConnections: currentConnectionsCount 0, maxConnectionsCount
-1.
[Sun Jan 30 23:02:56 2005] 000001e4 00000ec4 - TRACE: ws_server_group:
lockedServerGroupUseServer: Server app2Node_Web2b picked, weight 0.
[Sun Jan 30 23:02:56 2005] 000001e4 00000ec4 - TRACE: ws_server_group:
serverGroupIncrementConnectionCount: Server app2Node_Web2b picked,
pendingConnectionCount 1 totalConnectionsCount 87.
[Sun Jan 30 23:02:56 2005] 000001e4 00000ec4 - TRACE: ws_common:
websphereFindTransport: Finding the transport
[Sun Jan 30 23:02:56 2005] 000001e4 00000ec4 - TRACE: ws_common:
websphereFindTransport: Setting the transport(case 2): app2.itso.ibm.com on
port 9082

```

Overloading a cluster member

There are only so many concurrent requests that can be handled by a Web container in a cluster member. The number of concurrent requests is determined by the maximum number of threads available (10 threads implies 10 concurrent requests). However, a request does not necessarily constitute a user request. A browser might make multiple requests to get the information a user requested.

Connections coming into the Web container's WebContainer Inbound Chain feed requests to threads. If there are more connections than threads available, connections start to backlog, waiting for free threads. There is a maximum setting for the number of connections that can be backlogged as well. If the maximum number of connections in the backlog is exceeded, then no more connections will be allowed to the WebContainer Inbound Chain's port.

If there has been a successful connection but it is waiting for a thread in the Web container, then the plug-in will wait for a response (and so will the client). If the connection backlog is full, the plug-in will be refused a connection to the port and the plug-in will treat this in the same way as a stopped cluster member. It will

mark the cluster member as down and select a new cluster member to which to pass the request. The cluster member will then hopefully be able to reduce its connection backlog, since the plug-in will not try it again until the retry interval passes.

To avoid this overloading of cluster members, your environment needs to be configured to accept the load you are expecting. This includes the setting of weights that correspond to the system capabilities, the correct balance of cluster members and Web servers, and setting up the queues for requests and connections.

To monitor the behavior of the plug-in when a cluster member has too many requests, use a load testing tool (such as ApacheBench or JMeter), plug-in trace, and Tivoli Performance Viewer.

6.10.3 Tuning failover

There are a few places where failover can be tuned. This section details these settings. We cover the following options:

- ▶ Connection Timeout setting
- ▶ Retry interval setting
- ▶ Maximum number of connections

Connection Timeout setting

When a cluster member exists on a machine that is removed from the network (because its network cable is unplugged or it has been powered off, for example), the plug-in, by default, cannot determine the cluster member's status until the operating system TCP/IP timeout expires. Only then will the plug-in be able to forward the request to another available cluster member.

It is not possible to change the operating system timeout value without unpredictable side effects. For instance, it might make sense to change this value to a low setting so that the plug-in can fail over quickly.

However, the timeout value on some of the operating systems is not only used for outgoing traffic (from Web server to application server) but also for incoming traffic. This means that any changes to this value will also change the time it takes for clients to connect to your Web server. If clients are using dial-up or slow connections, and you set this value too low, they will not be able to connect.

To overcome this problem, WebSphere Application Server V6 offers an option within the plug-in configuration file that allows you to bypass the operating system timeout.

It is possible to change the connection timeout between the plug-in and each Application Server, which makes the plug-in use a non-blocking connect, as shown in Figure 6-19 on page 257. To configure this setting, go to **Application servers -> <AppServer_Name> -> Web Server plug-in properties**.

Setting the Connect Timeout attribute for a server to a value of zero (default) is equal to selecting the **No Timeout** option, that is, the plug-in performs a blocking connect and waits until the operating system times out. Set this attribute to an integer value greater than zero to determine how long the plug-in should wait for a response when attempting to connect to a server. A setting of 10 means that the plug-in waits for 10 seconds to time out.

To determine what setting should be used, you need to take into consideration how fast your network and servers are. Complete some testing to see how fast your network is, and take into account peak network traffic and peak server usage. If the server cannot respond before the connection timeout, the plug-in will mark it as down.

Since this setting is determined on the each Application Server, you can set it for each individual cluster member. For instance, you have a system with four cluster members, two of which are on a remote node. The remote node is on another subnet and it sometimes takes longer for the network traffic to reach it. You might want to set up your cluster in this case with different connection timeout values.

If a non-blocking connect is used, you will see a slightly different trace output. Example 6-21 shows what you see in the plug-in trace if a non-blocking connect is successful.

Example 6-21 Plug-in trace when ConnectTimeout is set

```
...
TRACE: ws_common: websphereGetStream: Have a connect timeout of 10; Setting
socket to not block for the connect
TRACE: errno 55
TRACE: RET 1
TRACE: READ SET 0
TRACE: WRITE SET 32
TRACE: EXCEPT SET 0
TRACE: ws_common: websphereGetStream: Reseting socket to block
...
```

Retry interval setting

There is a setting in the plug-in configuration file that allows you to specify how long to wait before retrying a server that is marked as down. This is useful in avoiding unnecessary attempts when you know that server is unavailable. The default is 60 seconds.

This setting is specified in the configuration of each Web server, as shown in Figure 6-24 on page 268, on the *Retry interval* field. This default setting means that if a cluster member was marked as down, the plug-in would not retry it for 60 seconds. To change this value, go to **Servers -> Web Servers -> <WebServer_Name> -> Plug-in properties -> Request Routing**.

There is no way to recommend one specific value; the value chosen depends on your environment, for example on the number of cluster members in your configuration.

For example, if you have numerous cluster members, and one cluster member being unavailable does not affect the performance of your application, then you can safely set the value to a very high number.

Alternatively, if your optimum load has been calculated assuming all cluster members to be available or if you do not have very many, then you will want your cluster members to be retried more often to maintain the load.

Also, take into consideration the time it takes to restart your server. If a server takes a long time to boot up and load applications, then you will need a longer retry interval.

Maximum number of connections

All requests to the application servers flow through the HTTP Server plug-in. The application server selection logic in the plug-in has been enhanced so that it takes into account the number of pending connections to the application server. The Maximum number of connections attribute is used to specify the maximum number of pending connections to an application server that can be flowing through a Web server process at any point in time.

Each application server can have a maximum number of pending connections coming from Web server plug-ins, as shown in Figure 6-19 on page 257. To change this setting, go to **Application servers -> <AppServer_Name> -> Web Server plug-in properties**.

The default setting is No Limit, which is the same as if the value is set to -1 or zero. The attribute can be set to any arbitrary value.

For example, let the two application servers be fronted by two nodes running IBM HTTP Server. If the MaxConnections attribute is set to 10, then each application server could potentially get up to 20 pending connections.

If the number of pending connections reaches the maximum limit of the application server, then it is not selected to handle the current request. If no other application server is available to serve the request, HTTP response code 503 (Service unavailable) is returned to the user.

This can be seen in the plug-in trace listing in Example 6-22.

Example 6-22 Plug-in trace when MaxConnections is set

```
..
### When the request comes, it is pended.
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0,
maxConnectionsCount 10.
TRACE: ws_server_group: lockedServerGroupUseServer: Server
wasInode_PluginMember1 picked, weight 0.
TRACE: ws_server_group: serverGroupIncrementConnectionCount: Server
wasInode_PluginMember1 picked, pendingConnectionCount 1 totalConnectionsCount
1.
TRACE: ws_common: websphereFindTransport: Finding the transport

### When the next request comes, it is pended again.
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 1,
maxConnectionsCount 10.
TRACE: ws_server_group: lockedServerGroupUseServer: Server
wasInode_PluginMember1 picked, weight 0.
TRACE: ws_server_group: serverGroupIncrementConnectionCount: Server
wasInode_PluginMember1 picked, pendingConnectionCount 2 totalConnectionsCount
2.
TRACE: ws_common: websphereFindTransport: Finding the transport

### When the number of pending connections reaches the MaxConnections(in this
case, 10), the HTTP response code 503 is returned to the user.

TRACE: ws_server_group: serverGroupGetFirstPrimaryServer: getting the first
primary server
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 10,
maxConnectionsCount 10.
STATS: ws_server_group: serverGroupCheckServerStatus: Checking status of
wasInode_PluginMember1, ignoreWeights 0, markedDown 0, retryNow 0, wlbAllows 1
reachedMaxConnectionsLimit 1
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 10,
maxConnectionsCount 10.
WARNING: ws_server_group: serverGroupCheckServerStatus: Server
wasInode_PluginMember1 has reached maximum connections and is not selected
```

```

TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next
primary server
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 10,
maxConnectionsCount 10.
STATS: ws_server_group: serverGroupCheckServerStatus: Checking status of
was2Node_PluginMember2, ignoreWeights 0, markedDown 0, retryNow 0, wlbAllows 1
reachedMaxConnectionsLimit 1
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 10,
maxConnectionsCount 10.
WARNING: ws_server_group: serverGroupCheckServerStatus: Server
was2Node_PluginMember2 has reached maximum connections and is not selected
ERROR: ws_server_group: serverGroupNextRoundRobinServer: Failed to find a
server; all could be down or have reached the maximum connections limit
WARNING: ws_common: websphereFindServer: Application servers have reached
maximum connection limit
ERROR: ws_common: websphereWriteRequestReadResponse: Failed to find a server
ERROR: ESI: getResponse: failed to get response: rc = 8
TRACE: ESI: esiHandleRequest: failed to get response
TRACE: ESI: esiRequestUrlStackDestroy
TRACE: ESI: esiRequestPopUrl: '/wlm/beenthere'
TRACE: ESI: esiUrlDestroy: '/wlm/beenthere'
ERROR: ws_common: websphereHandleRequest: Failed to handle request
TRACE: ws_common: websphereCloseConnection
TRACE: ws_common: websphereEndRequest: Ending the request
..

```

As mentioned earlier, when the plug-in detects that there are no application servers available to satisfy the request, HTTP response code 503 (Service unavailable) is returned. This response code appears in the Web server access log, as shown in Example 6-23.

Example 6-23 HTTP Server access log example

```

[08/Dec/2003:14:08:03 -0500] "GET /wlm/beenthere HTTP/1.0" 503 419
[08/Dec/2003:14:08:03 -0500] "GET /wlm/beenthere HTTP/1.0" 503 419
[08/Dec/2003:14:08:03 -0500] "GET /wlm/beenthere HTTP/1.0" 503 419

```

This feature helps you to better load balance the application servers fronted by the plug-in. If application servers are overloaded, the plug-in will automatically skip these application servers and try the next available application server.

This feature solves the main problem of application servers taking a long time to respond to requests. It is achieved by throttling the number of connections going to the application server through the plug-in.

EJB workload management

The Enterprise JavaBeans (EJB) specification is the foundation of the Java 2 Platform Enterprise Edition (J2EE). Vendors use this specification to implement an infrastructure in which EJBs can be deployed, and use a set of services such as distributed transactions, security, life cycle management, and database connectivity. IBM WebSphere Application Server Network Deployment V6.x provides another important service for EJBs: *workload management* (WLM).

We discuss the following topics in this chapter:

- ▶ Enabling EJB workload management
- ▶ EJB types and workload management
- ▶ EJB bootstrapping
- ▶ How EJBs participate in workload management
- ▶ EJB workload management routing policy
- ▶ EJB high availability and failover

Note: We have set up an identical copy of the ITSO sample topology described in Chapter 8, “Implementing the sample topology” on page 387 for this chapter.

This was done to be more independent while testing the failover scenarios, etc. We are using an identical environment but with different names (Web3, Web4a, and Web4b as the members of the WEBcluster and Ejb3, Ejb4a, and Ejb4b for the members of the EJBcluster) throughout this chapter.

7.1 Enabling EJB workload management

EJB workload management and high availability are achieved by a combination of WebSphere server cluster support and the WebSphere ORB (Object Request Broker) workload management (WLM) plug-in. This feature is provided by IBM WebSphere Application Server Network Deployment V6.x and is not available in WebSphere Application Server V6 or WebSphere Application Server - Express V6, since these are single server environments.

In IBM WebSphere Application Server Network Deployment V6, workload management for EJBs is enabled automatically when a cluster is created. No special configuration is needed to enable EJB WLM on the server. The WLM plug-in in the ORB takes the responsibility of dispatching the load among the application servers (cluster members) of the cluster.

Figure 7-1 shows how workload management is handled by IBM WebSphere Application Server Network Deployment V6.x.

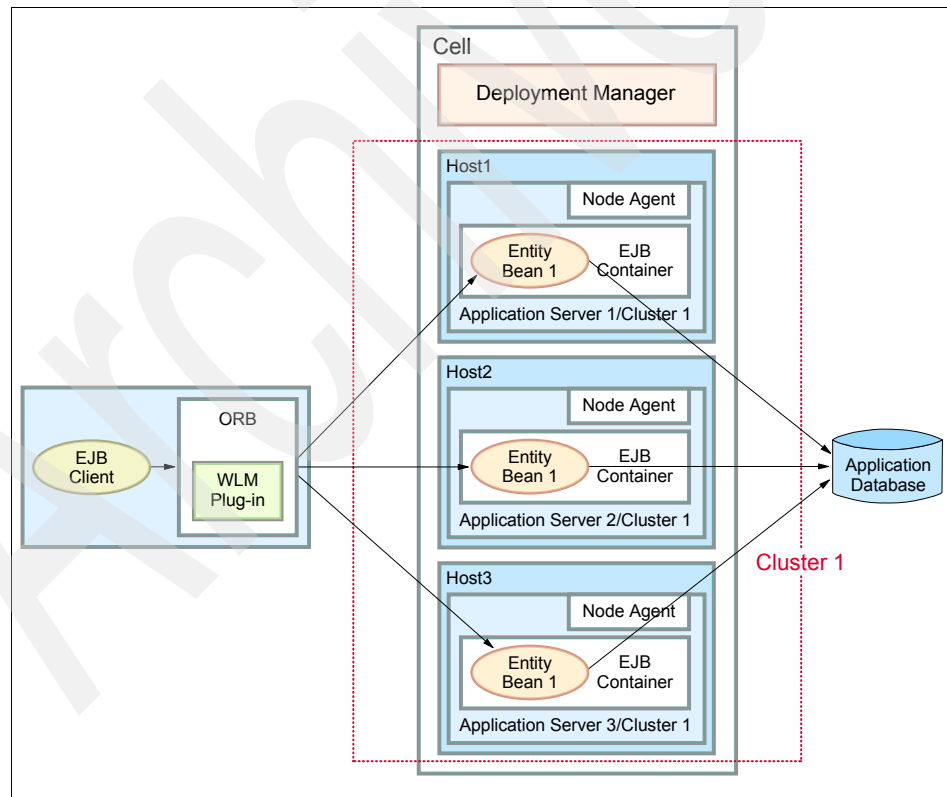


Figure 7-1 IBM WebSphere Application Server V6 EJB workload management

IBM WebSphere Application Server Network Deployment V6.x uses the concept of cell, cluster, and cluster members, as described in 1.3.3, “Workload management using WebSphere clustering” on page 19 to identify which application servers participate in workload management. Requests from clients are distributed among the cluster members’ EJB containers within the cluster.

The following types of EJB clients can participate in EJB WLM automatically:

- ▶ Clients in the same application server (servlets, JSPs, EJBs)
- ▶ Clients in a different application server (servlets, JSPs, EJBs)
- ▶ Java applications running within a WebSphere client container
- ▶ Stand-alone Java applications using the WebSphere-supplied Java Runtime Environment (JRE)

Note: The WebSphere-supplied Java Runtime Environment (JRE) is required for any remote Java clients because of WLM requirements for the WebSphere WLM-aware ORB.

7.2 EJB types and workload management

The workload management service provides load balancing and high availability support for the following types of EJBs:

- ▶ Homes of entity or session beans
- ▶ Instances of entity beans
- ▶ Instances of stateless session beans

As shown in Figure 7-2, the only type of EJB references not subject to load distribution through EJB WLM are stateful session bean instances. See 7.2.2, “Stateful session beans” on page 344 for details.

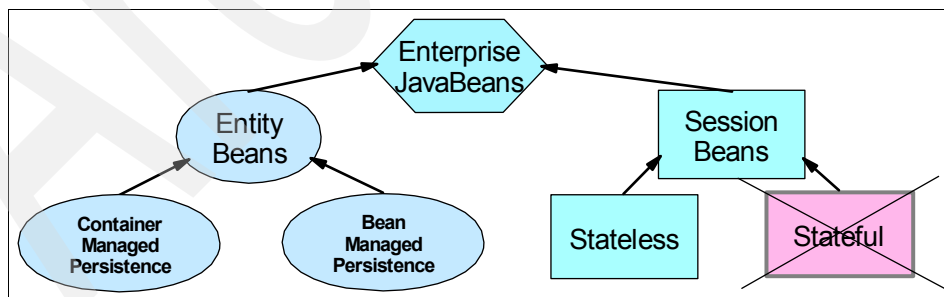


Figure 7-2 Enterprise beans that participate in workload management

7.2.1 Stateless session beans

By definition, a stateless session bean maintains no state information. Each client request directed to a stateless session bean is independent of the previous requests that were directed to the bean. The EJB container maintains a pool of instances of stateless session beans, and provides an arbitrary instance of the appropriate stateless session bean when a client request is received. Requests can be handled by any stateless session bean instance in any cluster member of a cluster, regardless of whether the bean instance handled the previous client requests.

Workload management can be applied to the Home object and the bean instance of a given stateless session bean. Therefore, the stateless session bean is a perfect programming model when constructing a well-balanced and highly available enterprise application.

7.2.2 Stateful session beans

A stateful session bean is used to capture state information that must be shared across multiple consecutive client requests that are part of a logical sequence of operations. The client must obtain an EJB object reference to a stateful session bean to ensure that it is always accessing the same instance of the bean.

WebSphere Application Server currently supports the clustering of stateful session bean home objects among multiple application servers. However, it does not support the clustering of a specific instance of a stateful session bean. Each instance of a particular stateful session bean can exist in just one application server and can be accessed only by directing requests to that particular application server. State information for a stateful session bean cannot be maintained across multiple application server cluster members. Thus, stateful session bean instances cannot participate in WebSphere workload management.

One significant improvement introduced in WebSphere Application Server V6 is the failover support for stateful session beans, which means that the state information maintained by a stateful session bean can survive various types of failures now. This is achieved by utilizing the functions of the Data Replication Service (DRS) and server workload management (WLM). More details on this topic can be found in 7.6, “EJB high availability and failover” on page 371.

Note: Even though stateful session beans are not workload-managed themselves, a certain level of WLM can be achieved when the homes are evenly distributed. It is only after the bean is created that everything will be performed on the same cluster member.

7.2.3 Entity beans

An entity bean represents persistent data. Most external clients access entity beans by using session beans, but it is possible for an external client to access an entity bean directly. The information contained in an entity bean is not associated with a session or with the handling of one client request or series of client requests. However, it is common for a client to make a succession of requests targeted at the same entity bean instance. It is also possible for more than one client to independently access the same entity bean instance within a short time interval. The state of an entity bean must therefore be kept consistent across multiple client requests.

For entity beans, the concept of a session is replaced by the concept of a transaction. An entity bean is instantiated in a container for the duration of the client transaction in which it participates. All subsequent accesses to that entity bean within the context of that transaction are performed against that instance of the bean in that particular container. The container needs to maintain state information only within the context of that transaction. The workload management service uses the concept of *transaction affinity* (as defined in 7.5.4, “Transaction affinity” on page 370) to direct client requests. After an EJB server entity bean is selected, client requests are directed towards it for the duration of the transaction.

Between transactions, the state of the entity bean can be cached. The WebSphere V6.x EJB container supports Option A, Option B, and Option C caching:

- Option A caching

WebSphere Application Server assumes that the entity bean is used within a single container. Clients of that bean must direct their requests to the bean instance within that container. The entity bean has exclusive access to the underlying database, which means that the bean cannot be clustered or participate in workload management if Option A caching is used.

- Option B caching

The bean instance remains active (so it is not guaranteed to be made passive at the end of each transaction), but it is always reloaded from the database at the start of each transaction. A client can attempt to access the bean and

start a new transaction on any container that has been configured to host that bean.

► Option C caching (default)

The entity bean is always reloaded from the database at the start of each transaction and passivated at the end of each transaction. A client can attempt to access the bean and start a new transaction on any container that has been configured to host that bean.

Entity beans can participate in workload management as long as the server reloads the data into the bean at the start of each transaction, assuming that transactional affinity is in place. Guaranteed passivation at the end of each transaction is not a requirement for a bean to participate in workload management. Hence, Option B and Option C caching are both compatible with workload management, but Option A caching is not.

Table 7-1 provides a summary of the EJB caching options.

Table 7-1 EJB caching options

Option	Activate at must be set to	Load at must be set to
A	Once	Activation
B	Once	At start of transaction
C (default)	At start of transaction	At start of transaction

Table 7-2 summarizes the workload management capability of different types of EJBs:

Table 7-2 Summary of EJB types and WLM

EJB types	Component	WLM-capable
Entity bean (Option A)	Home	Yes
	CMP bean instance	No
	BMP bean instance	No
Entity bean (Option B,C)	Home	Yes
	CMP bean instance	Yes
	BMP bean instance	Yes
Message-driven bean	Bean instance	Yes

EJB types	Component	WLM-capable
Session Bean	Home	Yes
	Stateless bean instance	Yes
	Stateful bean instance	No

7.3 EJB bootstrapping

In order to access EJBs deployed to WebSphere Application Server V6, the client, regardless of whether it is local or remote, must first obtain a reference to objects related to an application, such as a reference to an Enterprise JavaBean (EJB) home object. This process is called *EJB bootstrapping*. The bootstrapping service is provided through J2EE Naming that is implemented via WebSphere CORBA CosNaming.

EJB home objects are bound into a hierarchical structure, referred to as a *name space*. An InitialContext is used to access objects in the name space. To obtain an InitialContext, a bootstrap server and port need to be supplied. If these are not supplied, then default values are used specific to the client type and its environment.

InitialContext requests participate in workload management when the provider URL is a clustered resource (cluster member) and they do not when they are not a clustered resource.

More information about naming and name spaces can be found in the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451. Chapter 13 explains the concept in detail.

7.3.1 Bootstrapping within WebSphere containers

The bootstrapping process varies in looking up an object depending on whether or not the application is running in a WebSphere container. This section describes some methods and best practices of performing a JNDI lookup for EJB clients running within a WebSphere container environment.

Performing a lookup in an EJB or Web container in the same cell

When an application that is running within an EJB or Web container wants to perform a lookup of an EJB in the same cell, then the best practice is to use an

EJB reference in the application code and an InitialContext object with no parameters.

An EJB reference is a method of delegating the JNDI name resolution of an EJB from the application to the deployment descriptor. Using the prefix `java:comp/env/` before the JNDI name informs the container that this particular JNDI lookup resolves to a reference specified in the deployment descriptor. This indirect lookup removes the reliance on hard-coded JNDI names in the application code or reliance on external properties files.

Only when running in either a Web, EJB, or J2EE application client container can references be used since it is the job of the container to resolve those references using the deployment descriptors.

The binding of an EJB reference to a real JNDI name is specified in the deployment descriptor and can be altered during or after deployment using the WebSphere Administrative Console.

The InitialContext object used to perform the lookup does not need any parameters if the target EJB is in the same cell. Leaving the InitialContext object empty means that the local application server's naming service in which the client is running will be used for lookups. This is because all the name spaces throughout the cell are linked, so a fully qualified JNDI name will locate the EJB home.

If a lookup is performed from a Web or EJB container to an EJB that is in another process in the same cell, then the JNDI name needs to be either fully qualified with the node and server that contains the EJB or, if the EJB is on a clustered server, a JNDI name with the cluster name may be used. This means the JNDI name for `ejb/myEJB` can be one of the following:

- ▶ `cell/nodes/Node1/servers/Server1/ejb/MyEJB`
- ▶ `cell/clusters/MyCluster/ejb/MyEJB`

Example 7-1 shows a lookup of an EJB home when the client is running in an EJB or Web container that is looking up an EJB in the same cell. The EJB reference `java:comp/env/BeenThere` has been set to resolve to `cell/clusters/EJBcluster/BeenThere` in the EJB deployment descriptor.

Example 7-1 Lookup of an EJB home

```
// Get the initial context
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;

// Bootstrapping to get the InitialContext
```

```

Context initialContext = new InitialContext();

// Look up the home interface using the JNDI name
try {
    java.lang.Object ejbHome =
initialContext.lookup("java:comp/env/BeenThere");
    beenThereHome = (BeenThereHome)javax.rmi.PortableRemoteObject.narrow(
        (org.omg.CORBA.Object) ejbHome, BeenThereHome.class);
}
catch (NamingException ne) { // Error getting the home interface
    ...
}

```

From the sample code, the bootstrapping process of getting the JNDI InitialContext does not require the bootstrap server information, like server name and port number. Delegating the details of both the JNDI name and the InitialContext location to the container makes the application code much more portable among different environments.

“Server cluster with fault-tolerant initial context” on page 354 includes a discussion about fault tolerant name lookups; this does not apply here. If the local naming service is unavailable then the application server is probably not running.

Mapping EJB references to enterprise beans

With IBM WebSphere Application Server Network Deployment V6.x, you can bind the EJB references to enterprise beans during or after application deployment. System administrators can bind EJB references specified by the application developer to a required EJB home in a target operational environment by setting the JNDI name value using the Administrative Console as follows:

1. Select your application by clicking **Applications -> Enterprise Applications -> <application_name>**.
2. Select **Map EJB references to beans** from the Additional Properties section.

General Properties

* Name
BeenThere

Binary Management

* Application binaries
(APP_INSTALL_ROOT)/dmCell

☐ Use metadata from binaries

☒ Enable distribution

Validation
warn

Class Loading and File Update Detection

* Class loader mode
Parent First

* WAR class loader policy
Module

☒ Enable class reloading

Reloading interval
0

Additional Properties

- Stateful session bean failover settings
- Session management
- Application profiles
- Libraries
- Target mappings
- Last participant support extension
- View Deployment Descriptor
- Provide JMS and EJB endpoint URL information
- Publish WSDL files
- Provide HTTP endpoint URL information
- Map security roles to users/groups
- Provide JNDI Names for Beans
- Map EJB references to beans
- Map virtual hosts for Web modules
- Map modules to servers

Related Items

Figure 7-3 Map EJB references to beans

3. Enter the JNDI name of your Enterprise Bean(s) in the JNDI Name column as shown in Figure 7-4.

BeenThere > Mapping EJB references to enterprise beans

(EJB) reference that is defined in your application must map to an enterprise bean.

	Reference binding	Class	JNDI name
war,WEB-INF/web.xml	BeenThereBean	com.ibm.websphere.samples.beenthere.BeenThere	cell/clusters/EJBcluster/Bee

Figure 7-4 Map EJB references to enterprise beans - Enter JNDI name

4. Stop and restart all application server(s) where your application is installed.

Note: Mapping of EJB references to enterprise beans can also be performed at deployment time of your application.

Performing a lookup in an EJB/Web container outside the cell

The process is slightly different when performing a JNDI lookup from an EJB or Web container for an EJB that is not within the same cell. EJB references can still be used, but the local name server is not able to locate the EJB home. In this situation, a provider URL is needed when bootstrapping and creating the InitialContext object that points to the name server that contains the EJB naming information. This makes this scenario fall somewhere between a J2EE application client container and a stand alone client. EJB references can be used but the container cannot be relied on to locate a name server to find the EJB.

Performing a lookup in a WebSphere client container

The J2EE application client container provides similar facilities to an EJB or Web container in terms of naming. The client container performs EJB reference lookups so the code in Example 7-1 on page 348 can also be used in this case. It uses the application clients deployment descriptor to do the binding of a reference to a real JNDI name.

As the calling code is actually running outside of an application server process, it still needs to be able to resolve the InitialContext object to the correct location to look up the EJB. Using the J2EE application client container removes the need to hard-code the location of the application server that is providing the name service. The code in Example 7-1 on page 348 does a call without passing any parameters to the InitialContext. When running in the J2EE application client container, this is resolved to the JNDI name server that is specified when launching the client container. For example, using the command:

```
launchclient ClientApp.ear -CCBootstrapHost=app1  
-CCBootstrapPort=2809
```

will launch the J2EE client contained within the EAR file. When an InitialContext object is created with no parameters, the J2EE application client container resolves this to the name service running on app1 on port 2809.

This means that although the code is identical as for running in an EJB or Web container, the actual resolution of the correct name service relies on parameters passed to the J2EE application client container at initialization.

When running in this container, it is important that the client be able to resolve the process where the name server is running. If it is unable to do this then the client will fail to initialize. To get around this issue, it is possible to use a corbaloc provider URL to specify multiple locations for name servers that can be used to perform the JNDI lookup. This alternative command can be used:

```
launchclient ClientApp.ear  
-CCproviderURL=corbaloc::app1:9813,:app2:9814
```

When it comes to performing the InitialContext lookup, the container has a choice of name servers, either the application server on app1 listening on port 9813 or the application server on app2 listening on port 9814.

This list of locations is not processed in any particular order so there is no way to guarantee which server will be used, but keep in mind that this is just for looking up the EJB home object. Once that is obtained, normal EJB workload management decides which server in a cluster should actually be used. Should one of these name servers fail to respond then the other will be used, removing the single point of failure.

If the target EJB is running in a cluster then this is the recommended method of looking up EJB homes from a J2EE application client container as it provides failover. When populating this list, remember that whichever name server you point the client to has to be able to resolve the JNDI name the EJB reference is bound to. The possible options for fully qualifying the JNDI name are discussed at the end of “Performing a lookup in an EJB or Web container in the same cell” on page 347.

7.3.2 Bootstrapping outside of a J2EE container

Applications running outside of a container cannot use a naming reference like `java:comp/env` to look up names because only a J2EE container can configure the `java:name` space for an application. Instead, an application of this type must look up the object directly from the name server. Also, the stand-alone client cannot rely on a container to resolve the location of the name server as it is running outside of a container. Therefore, stand-alone Java clients have specific bootstrapping requirements; normally, the bootstrap server location needs to be specified in the application code programmatically.

Because the location of the name server and the JNDI name of the EJB are environment-specific, the stand-alone client should obtain the necessary values to pass to the InitialContext from an external resource, like a properties file.

In the following sections, we show three examples of JNDI lookups to application servers running the target EJB. These examples do not use a properties file. The samples demonstrate three scenarios where a stand-alone Java client bootstraps to a:

- ▶ Single server
- ▶ Server cluster
- ▶ Server cluster with fault-tolerant initial context

Single server

Example 7-2 on page 353 shows the lookup of an EJB home that is running in the single server, Server1, configured in the node app1. In this example, there is

only one server so there is no possibility of specifying multiple bootstrap servers. The name server is running on the same application server as the target EJB, so the JNDI name does not need to be fully qualified.

Example 7-2 Single server - lookup EJB home

```
// Get the initial context
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL, "corbaloc::app1:2809");
Context initialContext = new InitialContext(env);
// Look up the home interface using the JNDI name
try {
    java.lang.Object ejbHome = initialContext.lookup("BeenThere");
    beenThereHome = (BeenThereHome)javax.rmi.PortableRemoteObject.narrow(
        (org.omg.CORBA.Object) ejbHome, BeenThereHome.class);
}
catch (NamingException ne) { // Error getting the home interface
    ...
}
```

Server cluster

Example 7-3 shows the lookup of an EJB home that is running in a cluster called EJBcluster. The name can be resolved if any one of the cluster members is running. As this is a stand-alone client, the location of the name service still needs to be specified.

Example 7-3 Server cluster - lookup EJB home

```
// Get the initial context
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL, "corbaloc::app1:2809");
Context initialContext = new InitialContext(env);
// Look up the home interface using the JNDI name
try {
    java.lang.Object ejbHome = initialContext.lookup(
        "cell/clusters/EJBcluster/BeenThere");
    beenThereHome = (BeenThereHome)javax.rmi.PortableRemoteObject.narrow(
        (org.omg.CORBA.Object) ejbHome, BeenThereHome.class);
}
catch (NamingException e) { // Error getting the home interface
    ...
}
```

Server cluster with fault-tolerant initial context

In the previous example, the EJB is running on clustered servers but still relies on one server for the bootstrapping process. Example 7-4 shows how to obtain the initial context via a fault-tolerant provider URL which has bootstrapping information for two server cluster members:

Example 7-4 Cluster - lookup EJB home via fault-tolerant provider URL & specify path

```
// Get the initial context
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL, "corbaloc::app1:9811,:app2:9812");
Context initialContext = new InitialContext(env);

// Look up the home interface using the JNDI name
try {
    java.lang.Object ejbHome =
initialContext.lookup("cell/clusters/EJBcluster/BeenThere");
}
```

This is fault tolerant, so if one process goes down, another can be used as the naming provider, and the cluster can be accessed via the appropriate cell-based path in the lookup.

In the real world, application developers can further improve high availability and the workload management service for the stand-alone EJB client by adding HA and WLM logic to the above sample code. For example, developers can provide the code to dynamically control which bootstrap server should be used for the current client thread, controlling when to use the second bootstrap server, either through a round robin or random algorithm.

7.4 How EJBs participate in workload management

In this section, we examine how EJBs participate in workload management through the following stages:

- ▶ Initial request
- ▶ Subsequent requests
- ▶ Cluster run state changes

7.4.1 Initial request

When accessing an EJB, there are two groups of clients: cluster aware and cluster unaware. Cluster aware clients are those that are using an IBM ORB and therefore have access to the WLM information about WebSphere Application Server clusters. For more information about cluster aware and cluster unaware clients, see 7.5, “EJB workload management routing policy” on page 358.

These steps describe what happens when a cluster aware client accesses an EJB:

1. First, the client has to retrieve the initial context during the bootstrap process. This varies between client types as discussed in 7.3.1, “Bootstrapping within WebSphere containers” on page 347.
2. Next, the client needs to look up the EJB home based on the JNDI name, for example:

```
Object home = initContext.lookup("java:comp/env/BeenThere");
BeenThereHome beentherehome =
    (BeenThereHome) narrow(home, BeenThereHome.class);
```

Note: This example uses an EJB reference, `java:comp/env/BeenThere`. As discussed in “Performing a lookup in an EJB or Web container in the same cell” on page 347, this EJB reference must be bound to the fully qualified JNDI name of the deployed EJB, for example:

```
cell/clusters/EJBcluster/BeenThere
```

Using EJB references would not be possible in a stand-alone EJB client since it is not running in a container.

3. The client needs to create or retrieve an EJB object, using the create method or finders of the home interface, for example:

```
BeenThere beenThere = beentherehome.create();
```

4. Once the EJB object has been created, you can invoke methods from the remote interface, for example:

```
Hashtable envInfo = beenThere.getRuntimeEnvInfo();
```

We call these four steps the initial request from the EJB client. Let us see in detail what is happening from a workload management’s point of view, using Figure 7-5 on page 356:

1. The new InitialContext request goes through the ORB (Object Request Broker). This returns a JNDI context object which has the information of the server naming service.

2. The lookup on the context returns a home object of the BeenThere bean. This is an indirect IOR (Interoperable Object Reference), that is, it points to the Location Service Daemon (LSD) on the local Node Agent.
3. The first request goes to the LSD and the LSD selects one of the cluster members by using the WLM plug-in in the LSD.
4. The LSD returns a direct IOR to the specific cluster member.
5. The request is forwarded to the cluster member that the LSD selected.
6. Upon successful completion of the request, the response contains the cluster configuration information. The client WLM plug-in stores the cluster configuration information and uses it for subsequent requests.

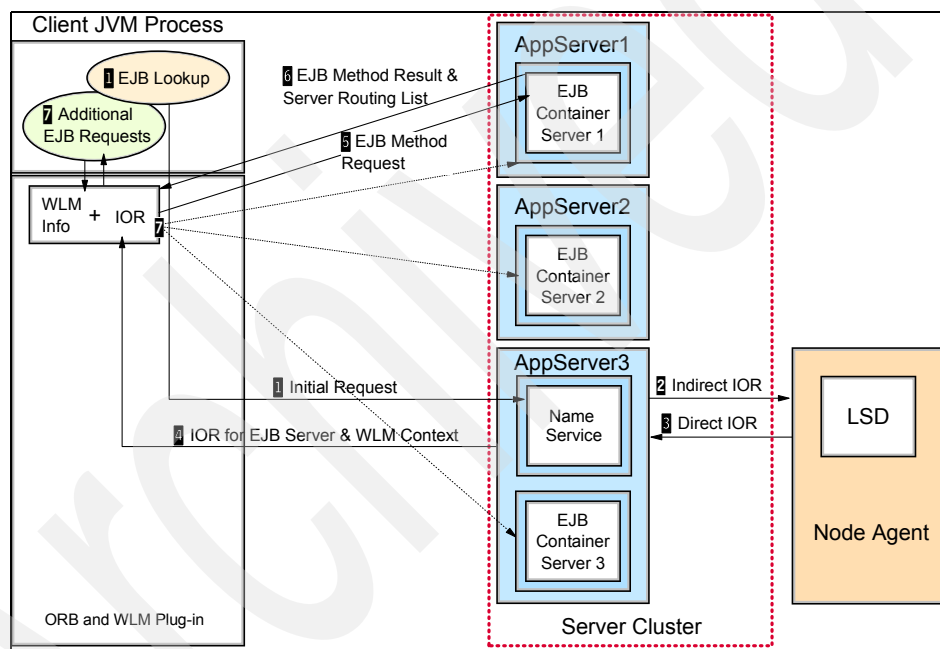


Figure 7-5 EJB workload management

7.4.2 Subsequent requests

Now let us look at what happens with subsequent EJB requests:

1. Subsequent requests from the client to a remote method go through the ORB as well.
2. The ORB asks the WLM plug-in for the IOR of the server in order to process the request. Generally, the workload decision is made here.

3. Based on the workload management policy, process affinity, and transaction affinity (see 7.5.1, “Server weighted round robin” on page 359), the WLM plug-in returns the IOR of the next target.
4. The ORB invokes the remote method on the selected server.

7.4.3 Cluster run state changes

In addition to the normal EJB request, let us take a look at how the WLM plug-in is informed of changes to the cluster configuration and dynamically propagates the change to WLM-aware clients:

1. Changes are made to the cluster configuration, such as adding and starting a fourth cluster member to the sample configuration.
2. The Deployment Manager pushes the changes to the Node Agents which in turn push those changes to all cluster members.
3. Meanwhile, the EJB client is still performing requests on the cluster.
4. With each request for a remote component, information about the cluster is returned in the response from the cluster member.

If the cluster has been modified since the last request from this client, the WLM plug-in updates its cluster data using the new information returned in the response.

5. The EJB client makes another method call to the remote interface.
6. The request handling ORB asks the WLM plug-in for the IOR of the server to contact.
7. The WLM plug-in returns the IOR of the next target, based on the workload management policy, process affinity, and transaction affinity (see 7.5.1, “Server weighted round robin” on page 359), then the request can be processed by the ORB.

Important: A change in the selection policy does not cause the cluster information to be sent to a client in response to a request. The WLM plug-in will continue using the selection policy defined at the first request. If the cluster topology or the number of active cluster members changes, the WLM plug-in will get the new selection policy as part of the new cluster configuration in the response.

Each Node Agent monitors the availability of the application servers running on its node. The Node Agent knows if an application server is still running by pinging it intermittently. If an application server fails then the Node Agent no longer receives responses on its ping messages and the Node Agent notifies the

Deployment Manager of the run state change. This information is then pushed out as was described in step 2 on page 357.

If a complete failure of a node occurs, then the Deployment Manager itself will not receive responses from its ping to the Node Agent and the clusters' configuration will be updated.

Important: One significant improvement in WebSphere Application Server V6 is that the Deployment Manager is no longer a single point of failure (SPOF) for WLM routing as in WebSphere V5. With the new High Availability Manager (HAMgr), a failure of the Deployment Manager triggers the HA Coordinator, which carries the WLM routing information, to fail over to any other server in the same HA core group, based on the defined HA policy. Thus, the WLM routing is a guaranteed service that is always available to the client even when a Deployment Manager failure occurs. This new HA management is discussed in Chapter 9, "WebSphere HAManager" on page 465.

7.5 EJB workload management routing policy

An EJB server routing policy defines how clients (such as servlets, stand-alone Java clients, or other EJBs) choose among EJB cluster members (instances). The server WLM relies on the routing policy to load balance or failover the client requests to available servers. The WebSphere workload management system provides a set of fairly complex routing policies capable of handling almost all kinds of workload balancing and failover requirements. WebSphere EJB workload management offers the following selection policies:

- ▶ Server weighted round robin
- ▶ Prefer local
- ▶ Process affinity
- ▶ Transaction affinity

We describe each routing policy in detail and these policies' relationships with the others in the following discussion. For each one of them, we explain the behavior of the EJB workload management under this selection policy and how to configure this type of routing policy in WebSphere.

We use the WebSphere sample application BeenThere to better demonstrate the EJB workload management behavior.

BeenThere is composed of a servlet and a stateless session EJB. The purpose of BeenThere is to collect information about the execution environment. The servlet retrieves information from the Web container, and the EJB obtains

information from the EJB container. The information retrieved is the server name, Web or EJB container name, and Java process ID.

7.5.1 Server weighted round robin

The server weighted round robin routing selects the next currently available cluster member based on a specific round robin algorithm. The policy ensures a distribution based on the set of server weights that have been assigned to the members of a cluster. For example, if all servers in the cluster have the same weight, the expected distribution for the cluster would be that all servers receive an equal number of requests. Otherwise, the distribution mechanism sends more requests to the higher weight value servers than the lower weight value servers. The policy ensures the desired distribution, based on the weights assigned to the cluster members. Weighted round robin policy greatly benefits an environment that has unbalanced hardware.

The server weight value defaults to 2 for each member of the cluster and is maintained in the cluster routing table.

Tip: When setting the server weight values for the cluster members, you should utilize low values to avoid load variations.

For example, you would be better off setting server weights to 2 and 5 versus 8 and 20 so that the refresh will occur more often and thus the server with the weight of 8 will not have to sit idle while 12 requests go to the server with a weight of 20. This way, it only sits idle for three requests instead of 12 requests.

Valid values for the weights range from 0 to 20.

If a particular EJB server instance is stopped or otherwise unavailable, that instance is skipped (no attempt is made to select it) until it can be verified as being back in service.

The ORB WLM plug-in in the EJB client maintains a routing table for each server cluster. The routing table is re-calculated for every new request coming in. There are no additional requests to an application server once its outstanding request ratio has reached its server weight value, but there are exceptions to this rule:

- Transaction affinity

Within a transaction, the first time an EJB server is chosen, the prevailing selection policy for the cluster member is applied. After the EJB server is selected, it remains bound for the duration of the transaction.

► Process affinity

This means that the EJB client and server are configured in the same application server. For example, the EJB container and Web container share the same application server and a servlet invokes an EJB, then the Web container never routes EJB requests to a container in another application server.

Note: Process affinity applies to servlet and EJB clients only. It does not affect stand-alone Java clients, because their EJB requests come from outside of the application server process. More on Transaction and Process affinity can be found in 7.5.3, “Process affinity” on page 370 and 7.5.4, “Transaction affinity” on page 370.

Let us take a close look at how server weighted round robin works in WebSphere Application Server V6. A server cluster has two cluster members as illustrated in Figure 7-6.

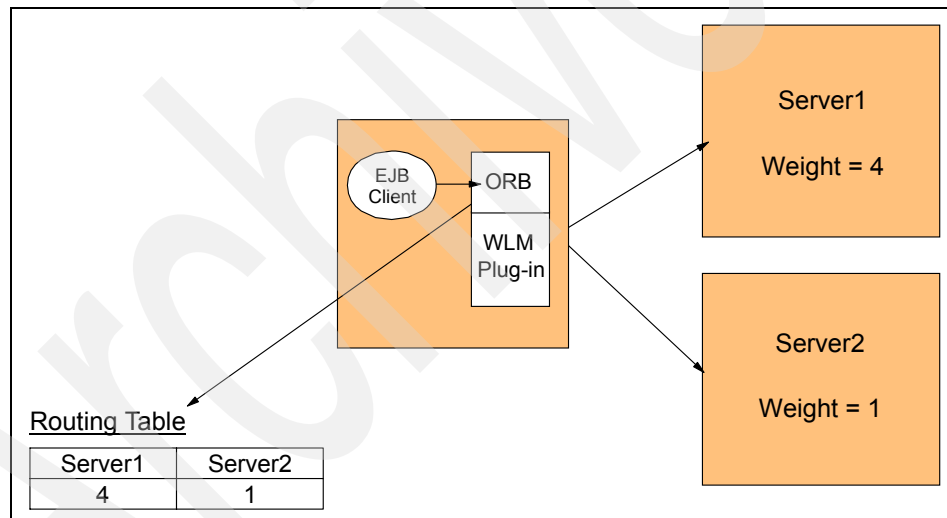


Figure 7-6 EJB WLM routing table for weighted round robin

Cluster member Server1 has a weight value of 4, another member, Server2, is weighted 1. After the bootstrapping process, the EJB client of this cluster gets the server clustering information, in particular the routing weight for each cluster member. Then, the ORB WLM plug-in creates and caches a WLM routing table reflecting the weight value of all available cluster members. In our example, this means 4 and 1 for Server1 and Server2 respectively. The selection decision of which cluster member to send the request to is based on the value provided by this table.

Table 7-3 describes how the routing table dynamically controls the client request workload management:

Table 7-3 Server weighted round robin routing table

After request	Server1 weight	Server2 weight
0 requests (initial state)	4	1
1. request	3	1
2. request	3	0
3. request	2	0
4. request	1	0
5. request	0	0
Reset:	4	1

1. Before the EJB client makes its first request, the routing table reflects the server weights: 4 and 1.
2. The first EJB request is routed to Server1 because of its higher weighted value. After this request, the weight for Server1 is decreased by 1 to 3 in the routing table; Server2 still has a weight of 1.
3. The second request goes round robin to Server2 and finds that its weight is greater than 0. Server2 is picked for this request, consequently, its weight is decreased to 0 after the second request.
4. After the third request, which was sent to Server1, the routing weights become 2 and 0.
5. Now, since Server2 has a weight of 0, all further requests are routed to Server1 until Server1's weight value also reaches 0 (for exceptions to this rule, please see the Note below).
6. Finally, the routing table has values of 0 and 0 for both servers. Now, the reset occurs and the initial weights are added back. Subsequent requests follow the same pattern to provide the EJB client a flexible workload management.

Notes:

- ▶ The routing table is decremented on each new request. No new requests are sent to an application server once its routing entry reaches zero or less, except when overridden by:
 - Affinity (Transaction, HTTPSession)
 - In Process (handled by ORB)
 - Prefer Local
- ▶ The original weights are added back even if a server has a weight below zero (for example because of transaction or process affinity). This might result in negative or zero weights for a server even after the reset. Please note that this behavior is different from that of the plug-in WLM (as described in 6.6.1, “Processing requests” on page 264) where the reset always results in positive weights because the plug-in adds multiples of the original weights if needed.
- ▶ WebSphere Application Server V6 introduces a new feature called “fairness balancing” for EJB WLM. For example, weights of 2 and 7 will result in a “a-bbbb-a-bbb” distribution rather than “a-b-a-bbbbbb”. This further reduces the possibility of load variations. However, using low weight values (for example 4 and 1 instead of 12 and 3) is still recommended.

Configuring server weighted round robin

Server weights are an attribute of the cluster, not the cluster member. They are associated with the members of a cluster and they are only meaningful when set on the cluster.

To set the EJB workload management server weights for your cluster members:

1. Locate the cluster member by selecting **Servers -> Clusters -> <cluster_name> -> Cluster members -> <member_name>** in the WebSphere Administrative Console.
2. On the Configuration tab, enter an appropriate value into the Weight field, as shown in Figure 7-7 on page 363.
3. Click **Apply** or **OK** and save the configuration changes.
4. Stop and start the cluster member(s) whose weight you have changed.

Server Cluster

[Server Cluster](#) > [EJBcluster](#) > [Cluster members](#) > **EJB4a**

An application server that belongs to a group of application servers called a cluster.

Runtime Configuration

General Properties

* Member name
EJB4a

* Weight
3

* Unique ID
1110381772340

Apply OK Reset Cancel

Figure 7-7 Workload management configuration weight value

Runtime changes

To make a change to a cluster's server weights so that only the current running system is affected, that is, so that the weights will not be saved to the cluster's configuration in the configuration repository, do the following:

1. Locate the cluster member by clicking **Servers -> Clusters -> <cluster_name> -> Cluster members -> <member_name>** in the WebSphere Administrative Console.
2. Select the **Runtime** tab and enter the right value into the Weight field.

Tip: If you do not see the Runtime tab, your server is down.

3. Click **Apply** or **OK**.

The changes become active as soon as you click **Apply** or **OK**. Restarting the cluster or application server reloads the values from the configuration.

Example: WLM behavior using server weighted round robin

Now we use the BeenThere application to demonstrate the WLM behavior of server weighted round robin.

We use the topology shown in Figure 7-8 to demonstrate the WLM policies. For performance reasons, this topology is not recommended for use in a production environment. We use this configuration only to show the various EJB workload management features.

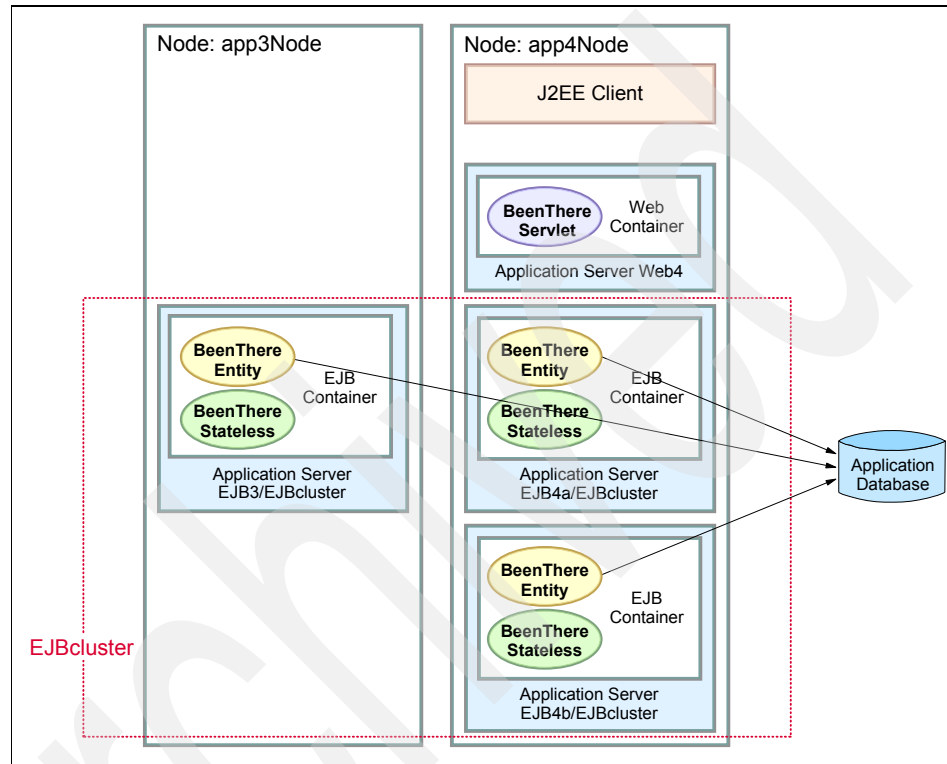


Figure 7-8 Sample topology for EJB workload management policies

Table 7-4 lists the cluster members weight values used in our configuration:

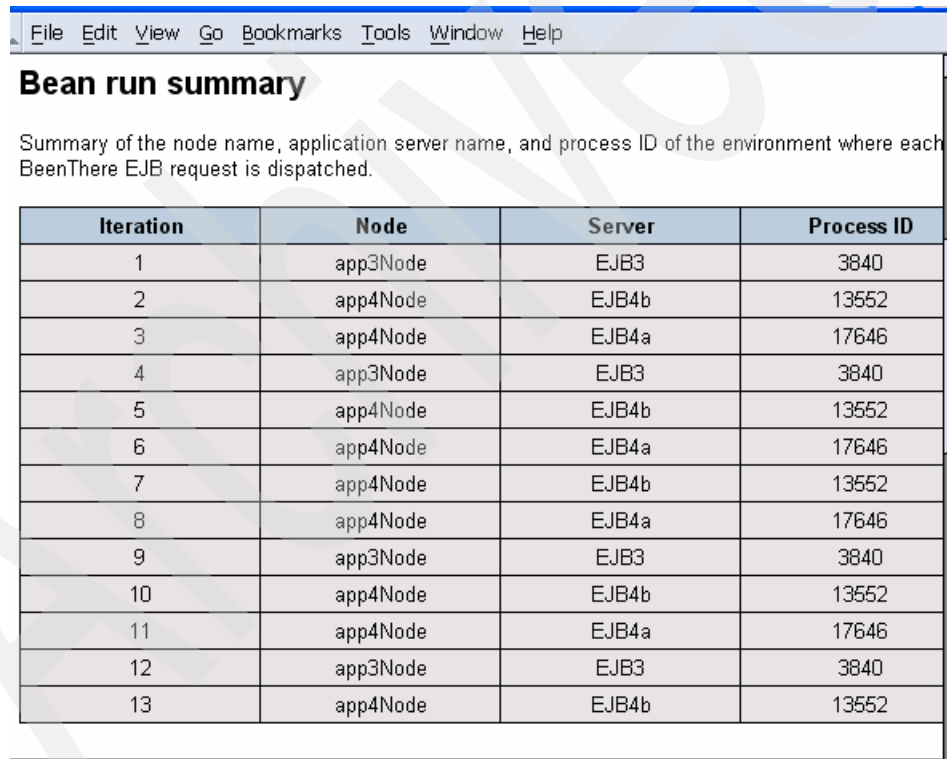
Table 7-4 Weight values used for our examples

Cluster member	Weight
EJB3	2
EJB4a	3
EJB4b	3

Assuming a request has no affinity, the server weighted round robin routing will select the next currently available cluster member, based on the weights of the cluster members.

Important: Whatever the server selection policy, process affinity and transaction affinity always override the selection policy.

For example, in Table 7-4 on page 364, the cluster weights describe a 2/8, 3/8, 3/8 request distribution. This means that if there are eight requests sent from a client, EJB3 will get two of the requests, while EJB4b and EJB4a each get three requests. As you can see in Figure 7-9, the EJB requests are distributed across all available cluster members in a repetitive order until the value of the lower weighted cluster member EJB3 reaches zero, at which point EJB4b and EJB4a receive an extra request each. Remember, both EJB4a and EJB4b have the same server weight value of 3, while EJB3 has a server weight value of 2. After eight requests, WLM plug-in resets the routing table, it is back to 2 - 3 - 3. The routing cycle starts all over again based on the same pattern.



Iteration	Node	Server	Process ID
1	app3Node	EJB3	3840
2	app4Node	EJB4b	13552
3	app4Node	EJB4a	17646
4	app3Node	EJB3	3840
5	app4Node	EJB4b	13552
6	app4Node	EJB4a	17646
7	app4Node	EJB4b	13552
8	app4Node	EJB4a	17646
9	app3Node	EJB3	3840
10	app4Node	EJB4b	13552
11	app4Node	EJB4a	17646
12	app3Node	EJB3	3840
13	app4Node	EJB4b	13552

Figure 7-9 Web client calling EJBs - round robin server selection

7.5.2 Prefer local

Besides the server weighted round robin routing, WebSphere Application Server V6 provides another WLM routing policy: Prefer local.

This policy has a global scope for a given cluster. Once the Prefer local policy is turned on, it is applied for every cluster member in the cluster. Similarly, when you turn it off, it is off for every cluster member in your cluster.

With the Prefer local policy, the selection made by the WLM plug-in not only depends on the running cluster members and their weight, but also on the node where the request comes from. The WLM plug-in will only select cluster members on the *same* node as the client, unless all local cluster members are unavailable.

The advantage of the Prefer local policy is that there is no network communication between the client and the EJB, so depending on the chosen topology, this policy can significantly improve the overall performance.

The client is the Java virtual machine (JVM) in which the code calling the EJB is running. This client might be a WebSphere process such as an application server running a Web container, or an application server running an EJB container.

Prefer local results in the following behaviors:

- ▶ A servlet calls an EJB: the request goes to the EJB container running on the same system (node).
- ▶ EJB1 is calling EJB2: the request goes to the same EJB container if EJB2 can be found (see 7.5.3, “Process affinity” on page 370), or to another EJB container running on the same node.
- ▶ In the case of a Java program running on the same machine as WebSphere and using the WebSphere JRE and its ORB, the Prefer local policy will dispatch requests among EJB containers running on the same machine. EJB WLM requires the WebSphere ORB and its WLM plug-in. If non-WebSphere ORBs are used, then the Java client will not be able to participate in EJB WLM.

The client accessing an EJB may also be a Java virtual machine not running in WebSphere, such as a J2EE client application, or a stand-alone Java program accessing EJBs. For a remote J2EE client application, the Prefer local policy has no influence on the request distribution because the client runs on a remote workstation.

If there is no cluster member available on the local system (because of a failure, or because of the topology), the request will be dispatched to available cluster

members following the server weighted round robin routing policy, as described in 7.5.1, “Server weighted round robin” on page 359.

The selection of the Prefer local option should be based on topology and pre-production test results. Naturally, the local host call will be quicker, and if you can put your clients (usually Web containers) on the same machines as your servers, the Prefer local option is a good choice. If you have clients on a subset of your machines, then you should analyze the load distribution, since client requests come from remote machines as well as from the local machine.

Configuring the Prefer local routing policy

To activate the EJB workload management Prefer local option for a cluster:

1. Select the cluster by clicking **Servers -> Clusters -> <cluster_name>** in the WebSphere Administrative Console.
2. Check the **Prefer local** box on the Configuration tab as shown in Figure 7-10.
3. Click **Apply** or **OK** and save the configuration changes.
4. Stop and start the changed cluster member(s) to activate Prefer local in their configuration.

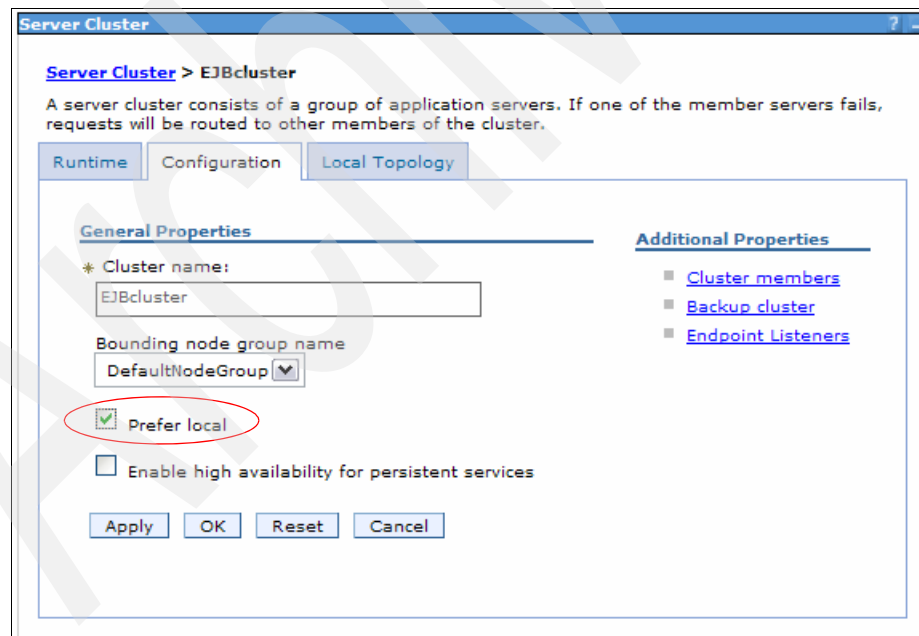


Figure 7-10 Workload management configuration Prefer local option

Runtime changes

To activate the EJB workload management Prefer local option so that it only affects the currently running system, that is, the Prefer local activation is not saved to the cluster configuration in the configuration repository, do the following:

1. Select the cluster by clicking **Servers -> Clusters -> <cluster_name>** in the WebSphere Administrative Console.
2. Select the **Runtime** tab.
3. Check the **Prefer local** box and click **Apply** or **OK**.

The change takes effect immediately.

Example: WLM behavior using Prefer local

We again use the BeenThere application to demonstrate the EJB workload management behavior under the Prefer local routing policy.

As mentioned previously, in the case of a Java program running on the same machine as WebSphere and using the WebSphere JRE and its ORB, the Prefer local policy will dispatch requests among EJB containers running on the same node. EJB WLM requires the WebSphere ORB and its WLM plug-in; clients using non-WebSphere ORBs cannot participate in EJB WLM.

Important: Whatever the server selection policy, process affinity and transaction affinity always override the selection policy.

Our requests were sent to node app4Node, which hosts the Web application (Web4a or Web4b) as well as EJB4a and EJB4b. As you can see in Figure 7-11 on page 369, EJB requests are distributed across the available cluster members on the local node, app4Node, in a repetitive order. Remember, both EJB4a and EJB4b have the same server weight value of 3.

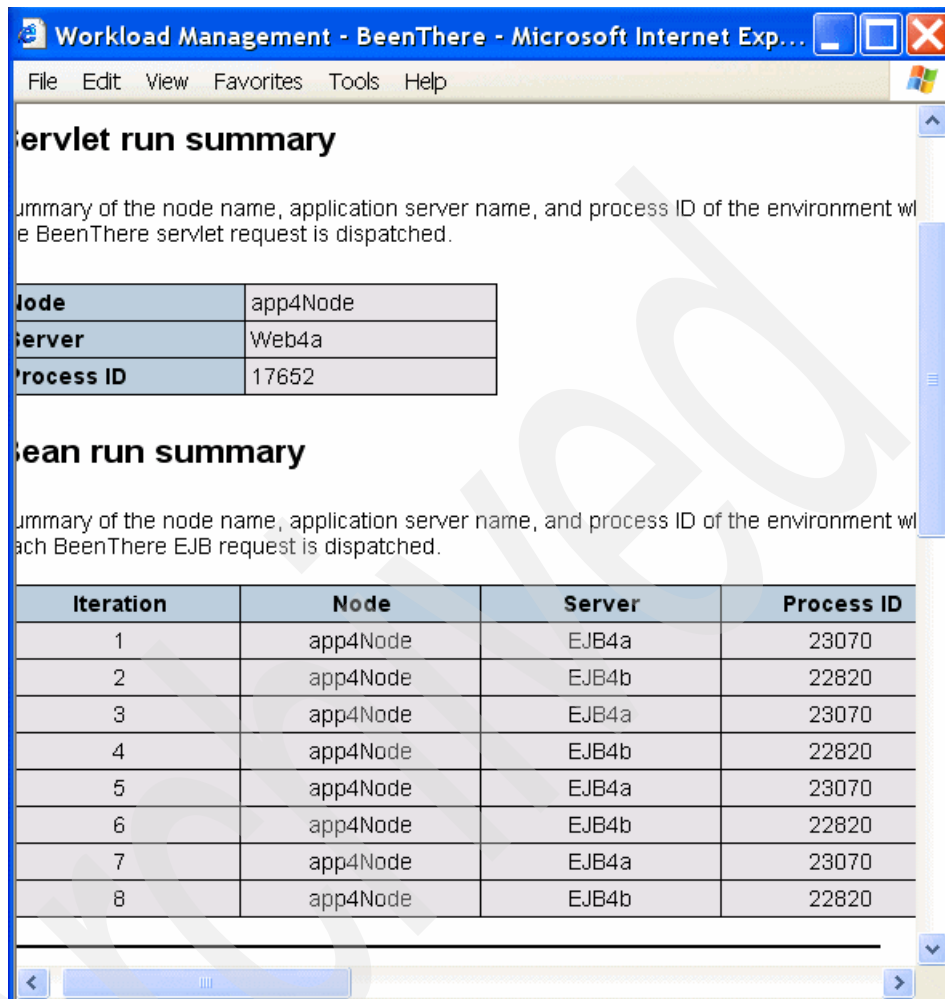


Figure 7-11 Web client calling EJBs - Prefer local policy enabled

As you can see, the behavior is different than for the server weighted round robin routing shown in Figure 7-9 on page 365. The client request is never routed to cluster member EJB3, which is deployed to app3Node. The Prefer local policy guarantees that the request is routed to the cluster members on the same node.

7.5.3 Process affinity

Regardless of the workload management server selection policy, if an EJB is available in the same cluster member as the client, all requests coming from that client are directed to the EJB in the same JVM process. This is called *process affinity*, because all requests are in-process requests. The advantage of process affinity is that there is no need for serialization for method calls. Parameters can be passed by value without any serialization costs, since all method calls are performed within the same Java Virtual Machine, in the same memory space.

To take advantage of process affinity, the client can only be a servlet or an EJB. In the case of a servlet, process affinity is only possible if the Web container running the servlet is in the same application server as the EJB container. In the case of an EJB (a stateless session bean acting as a facade, for instance, as recommended by EJB development best practices), process affinity occurs when the called EJB is in the same EJB container as the calling EJB. With IBM WebSphere Application Server Network Deployment V6, you can only have one EJB container per application server.

Note: Process affinity overwrites the Prefer local policy. See 7.5.2, “Prefer local” on page 366 for more information about this policy.

7.5.4 Transaction affinity

When several requests to EJB methods occur in the scope of the same transaction (a user-managed transaction or a container-managed transaction), all requests will go to the same cluster member, if possible. As soon as the WLM plug-in notices that a transaction is started, it stops dispatching the requests to different cluster members. All requests within the scope of this transaction are sent to the same cluster member.

Note: Transaction affinity overwrites all other server selection policies.

7.6 EJB high availability and failover

Many J2EE applications rely on Enterprise JavaBeans (EJBs) to implement key business logic. Therefore, providing a resilient and highly available EJB runtime system is a critical task for any EJB container provider. WebSphere Application Server V6 satisfies this requirement for EJB applications by providing an advanced high availability (HA) solution which guarantees that EJB requests can be serviced continuously even during various types of failures.

The EJB HA solution is not an isolated implementation; instead, it is part of a broader scope of HA solutions provided by WebSphere as core services to almost all kinds of J2EE components running in WebSphere. Please refer to Chapter 9, “WebSphere HAManager” on page 465 for details.

When an EJB client makes calls from within the WebSphere container, client container or outside of a container, the request is handled by the EJB container in one of the cluster members. If that cluster member fails, the client request is automatically redirected to another available server. In IBM WebSphere Application Server Network Deployment V6, the EJB HA is achieved by a combination of three WebSphere services: the HAManager, the EJB server cluster and EJB workload management (WLM).

7.6.1 EJB client redundancy and bootstrap failover support

When planning an EJB HA solution, not only is EJB server redundancy needed, but EJB client failover and redundancy should also be considered. EJB client redundancy refers to the automatic failover capability for an EJB request originator. In other words, an end user that initiated an EJB request can recover from the failure of a particular EJB client instance.

The first task for any EJB client is to look up the home of the bean (except MDB). The following two scenarios need to be considered:

- EJB requests coming from a clustered environment

Examples could be Web clients from Web containers that are workload-managed, EJB clients from another EJB container server cluster, or EJB clients from their own server cluster. In this case, EJB clients can use their own server to bootstrap with the default provider URL. If the bootstrap fails, the EJB client fails. This failure should be handled by the previous server, for example the Web server plug-in. Another version of the same client in a different container may bootstrap from its server successfully. By using client redundancy, EJB failover and high availability can be achieved.

- EJB requests coming from a non-clustered environment

Examples could be a Java client, J2EE client, C++ client, or third-party ORB client. In this case, if the client is bootstrapping to only one server, the client fails if the server fails, since the client is not redundant. You should bootstrap the client to as many bootstrap servers as possible. This can be achieved by using the bootstrapping workload management as discussed in 7.3, “EJB bootstrapping” on page 347.

Example 7-5 Lookup with more than one bootstrap server

```
prop.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");
prop.put(Context.PROVIDER_URL, "corbaloc::host1:9810,host2:9810");
Context initialContext = new InitialContext(prop);
try { java.lang.Object myHome = initialContext.lookup("MyEJB");
```

From the above example, the EJB client has the information for two bootstrap servers. Therefore, if the request to server host1 fails, the bootstrap engine will automatically redirect the bootstrap request to the server on host2.

WebSphere Application Server V6 uses the CORBA CosNaming as a naming solution. It is often convenient to use multiple bootstrap addresses for automatic retries. Every WebSphere server process contains a naming service, and the client application can bootstrap to any combination of servers. It is a good practice to bootstrap to the servers in a cluster since the InitialContext is workload-managed and you can use the simple name in the lookup. The other option is to get the InitialContext directly from the Node Agent using the default port (usually 2809) and use the fully qualified name in the lookup. This option, however, is not recommended, because the Node Agent is not workload-managed and has no failover capability.

For J2EE clients, you can specify a bootstrap host and port in the launch command line, and try different hosts/ports if you do not succeed when using the default URL provider.

Once the EJB home is successfully looked up, the naming server returns an indirect IOR, LSD, and routing table, and the WLM plug-in redirects the client to one of the clustered EJB containers.

7.6.2 EJB container redundancy and EJB WLM failover support

High availability of the EJB container is achieved using a combination of the WebSphere server cluster support and workload management plug-in to the WebSphere ORB.

Redundant server topology for EJB failover support

As shown in Figure 7-12, horizontal and vertical scaling is used for application server redundancy to tolerate possible process and machine failures.

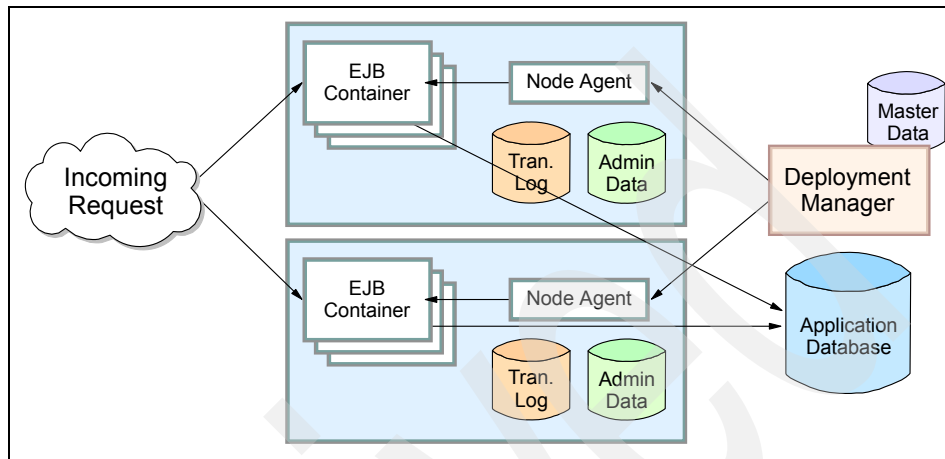


Figure 7-12 WebSphere EJB container failover

The mechanisms for routing workload-managed EJB requests to multiple cluster members are handled on the client side of the application. In WebSphere Application Server V6, this functionality is supplied by a workload management plug-in to the client ORB and the routing table in the LSD hosted in the Node Agent. The WLM failover support for EJBs is to maintain the routing table and to modify the client ORB to redirect traffic in case of a server failure.

The following is a list of possible failures for WebSphere processes:

- ▶ Expected server process failures, for example, stopping the server.
- ▶ Unexpected server process failures, for example, the server JVM crashes.
- ▶ Server network problems, for example, a network cable is disconnected or a router is broken.
- ▶ Machine problems, for example, a system shutdown, operating system crashes, or power failures.
- ▶ Overloading of EJB clients, for example, a denial of service attack, where the system is not robust enough to handle a large number of clients, or the server weight is inappropriate.

7.6.3 EJB failover behavior

For a specific WebSphere environment, the following three types of servers can fail:

- ▶ Deployment Manager
- ▶ Node Agent
- ▶ WebSphere Application Server cluster member

Deployment Manager failure

The Deployment Manager provides runtime support for WLM services, which is the key to a successful EJB failover. If the Deployment Manager fails, the WLM information can no longer be propagated to associated cluster members and EJB clients. This used to be a major concern in WebSphere Application Server V5.x because it presented a single point of failure (SPOF) for EJB workload management.

In IBM WebSphere Application Server Network Deployment V6, this limitation has been eliminated by the introduction of the WebSphere High Availability service. A new WebSphere component, the HAManager, is responsible for running key services like WLM on all available servers rather than on a single dedicated Deployment Manager. Once a Deployment Manager failure is detected, the HAManager quickly delegates the WLM service to another available server, such as one of the Node Agents or application servers. This provides continuous service for EJB workload management. Therefore, the Deployment Manager is no longer a SPOF for EJB workload management, which is a significant improvement for WebSphere high availability.

Please refer to Chapter 9, “WebSphere HAManager” on page 465 for more information about the HAManager functionality.

Node Agent failure

The Node Agent provides several important services to the Deployment Manager, application servers, and application clients. Among these services, we are most interested in the Location Service Daemon (LSD) service which is used by EJB workload management to provide the WLM routing information to clients.

If a Node Agent failure occurs after the routing table is available on the client, the WLM-enabled client code does not need to go to the LSD to determine to which server the request should be routed. The WLM-aware client code handles the routing decisions.

However, if the failure occurs before the first client request can retrieve the WLM information, then WLM depends on the LSD request to fail over to another LSD. Since there is no automatic failover of this service (or the Node Agent) in

WebSphere V6, the developer should make sure that the client has several options (servers) to retrieve the WLM information. See 7.3.2, “Bootstrapping outside of a J2EE container” on page 352 for information about how this can be achieved.

Cluster member failure

If the failure occurs on the first initial request where the routing table information is not yet available, a `COMM_FAILURE` exception is returned and the ORB recognizes that it has an indirect IOR available and resends the request to the LSD to determine another server to route to. If the failure occurs after the client retrieves the routing table information, the WLM client handles the `COMM_FAILURE`. The server is removed from the list of selectable servers and the routing algorithm is used to select a different server to route the request to.

Consider the following sequence of a client making a request to the EJB container of an application server:

1. For the initial client request, no server cluster and routing information is available in the WLM client's runtime process. The request is therefore directed to the LSD that is hosted on a Node Agent to obtain routing information. If the LSD connection fails, the request is redirected to an alternative LSD if specified in the provider URL. If this is not the first request, the WLM client already has routing information for WLM-aware clients. For future requests from the client, if there is a mismatch of the WLM client's routing information with what is on a server's, new routing information is added to the response (as service context). However, for WLM-unaware clients, the LSD always routes requests to available servers.
2. After getting the `InitialContext`, the client does a lookup to the EJB's home object (an indirect IOR to the home object). If a failure occurs at this time, the WLM code transparently redirects this request to another server in the cluster that is capable of obtaining the bean's home object.
3. A server becomes unusable during the life cycle of the request:
 - If the request has strong affinity, there cannot be a failover of the request. The request fails if the original server becomes unavailable. The client must perform recovery logic and resubmit the request.
 - If the request is to an overloaded server, its unresponsiveness makes it seem as though the server is stopped, which may lead to a time-out. Under these circumstances, it may be helpful to change the server weight and/or tune the ORB and pool properties:
 - `com.ibm.CORBA.RequestTimeout`
 - `com.ibm.CORBA.RequestRetriesCount`
 - `com.ibm.CORBA.RequestRetriesDelay`
 - `com.ibm.CORBA.LocateRequestTimeout`

These properties can be changed using the command line or the Administrative Console.

- If a machine becomes unreachable (network and/or individual machine errors) before a connection to a server has been established, the operating system TCP/IP keep-alive time-out dominates the behavior of the system's response to a request. This is because a client waits for the OS-specific, keep-alive time-out before a failure is detected. See “Connection Timeout setting” on page 335 for additional information.
- If a connection is already established to a server, `com.ibm.CORBA.RequestTimeout` is used (the default value is 180 seconds), and a client waits this length of time before a failure is announced. The default value should only be modified if an application is experiencing time-outs repeatedly. Great care must be taken to tune it properly. If the value is set too high, failover may become very slow; if it is set too low, requests may time out before the server has a chance to respond.

The two most critical factors affecting the choice of a time-out value are the amount of time to process a request and the network latency between the client and server. The time to process a request, in turn, depends on the application and the load on the server. The network latency depends on the location of the client. For example, those running within the same LAN as a server may use a smaller time-out value to provide faster failover. If the client is a process inside of a WebSphere Application Server (the client is a servlet), this property can be modified by editing the request time-out field on the Object Request Broker property panel. If the client is a Java client, the property can be specified as a runtime option on the Java command line, for example:

```
java -Dcom.ibm.CORBA.RequestTimeout=<seconds> MyClient
```

- A failed server is marked unusable, and a JMX notification is sent. The routing table is updated. WLM-aware clients are updated during request/response flows. Future requests will not route requests to this cluster member until new cluster information is received (for example, after the server process is restarted), or until the expiration of the `com.ibm.websphere.wlm.unusable.interval`. This property is set in seconds. The default value is 300 seconds. This property can be set by specifying `-Dcom.ibm.websphere.wlm.unusable.interval=<seconds>` as a command-line argument for the client process.

EJB failover depends on whether or not this type of EJB can be workload-managed by the container. Basically, WebSphere Application Server provides HA and failover support for all WLM-capable EJB types. The WLM-capable EJB types are listed in 7.2, “EJB types and workload

management” on page 343. Based on that, Table 7-5 summarizes the failover capability for the various EJB types.

Table 7-5 Summary of EJB types and failover support

EJB types	Component	Failover capable
Entity bean (Option A)	Home	Yes
	CMP bean instance	No
	BMP bean instance	No
Entity bean (Option B,C)	Home	Yes
	CMP bean instance	Yes
	BMP bean instance	Yes
Session Bean	Home	Yes
	Stateless bean instance	Yes
	Stateful bean instance	Yes

Stateful session beans are an exception to the fact that the failover capability of EJBs depends on the WLM capability of that EJB type (compare Table 7-1 on page 346). The failover of stateful session beans is a new feature in IBM WebSphere Application Server Network Deployment V6. See “Stateful session bean failover support” on page 377 for more information. Therefore, IBM WebSphere Application Server Network Deployment V6 supports the failover for almost all types of Enterprise JavaBeans.

Stateless session bean failover support

The EJB container maintains a pool of instances of stateless session beans and provides an arbitrary instance of the appropriate stateless session bean when a client request is received. Requests can be handled by any stateless session bean instance in any cluster member, regardless of whether the bean instance handled the previous client requests. If an EJB cluster member fails, the client request can be redirected to the same stateless EJB deployed under another WebSphere Application Server cluster member, based on the WLM routing policy.

Stateful session bean failover support

IBM WebSphere Application Server Network Deployment V6 supports stateful session bean failover in the event of unexpected server failures. This is achieved by using WebSphere Data Replication Service (DRS) and Workload

Management (WLM) functions. For more details about DRS, see 6.8.7, “Understanding DRS (Data Replication Services)” on page 297.

Unlike the failover support for stateless session beans, the highly available stateful session bean does not utilize a redundant array of stateful session bean instances, but rather replicates its state in a highly available manner such that when an instance fails, the state can be recovered and a new instance can take the failed instance's place. The state replication of a stateful session bean to another instance is handled by DRS.

Stateful session bean failover is provided by WebSphere as a runtime feature. You can use the WebSphere Administrative Console to enable or disable the failover support. Depending on the scope of the failover target, you can enable or disable the stateful session failover at the following three levels:

- ▶ EJB container
- ▶ Enterprise application
- ▶ EJB module

This feature provides great flexibility to end users under different conditions, for example:

- ▶ If you want to enable failover for all applications except for a single application, enable failover at the EJB container level and override the setting at the application level to disable failover for the single application.
- ▶ If you want to enable failover for a single installed application, disable failover at the EJB container level and then override the setting at the application level to enable failover for the single application.
- ▶ If you want to enable failover for all applications except for a single module of an application, enable failover at the EJB container level, then override the setting at the module application level to disable failover for the single module.
- ▶ If you want to enable failover for a single installed EJB module, disable failover at the EJB container level and then override the setting at the EJB module level to enable failover for the single EJB module.

Now, let's look at how to enable the failover support for stateful session beans at the three different levels.

Enabling/disabling stateful session bean failover at the EJB container level

1. In the Administrative Console, select **Servers -> Application servers -> <AppServer_Name>**.
2. Expand **EJB Container Settings** then select **EJB container**.

3. Select the **Enable stateful session bean failover using memory-to-memory replication** checkbox. See Figure 7-13.

This checkbox is disabled until you define a replication domain. The selection has a hyperlink to help you configure the replication settings. If no replication domains are configured, the link takes you to a panel where you can create one. If at least one domain is configured, the link takes you to a panel where you can select the replication settings to be used by the EJB container.

4. Click **OK** and save your changes.

Application servers > EJB4a > EJB container

An EJB container is a component of a J2EE application server that provides runtime services to ejb modules which can be deployed within it.

Configuration

General Properties

- * Passivation directory:
- Inactive pool cleanup interval: milliseconds
- Default data source JNDI name:
- ☒ Enable stateful session bean failover using [memory-to-memory replication](#)
- Initial State:

Additional Properties

- [EJB cache settings](#)
- [EJB timer service settings](#)

Figure 7-13 Configure stateful session bean failover at EJB container level

Enabling/disabling stateful session bean failover at the application level

1. In the Administrative Console, select **Applications -> Enterprise Applications -> <Application_name>**.
2. Select **Stateful session bean failover settings** from the Additional Properties.
3. Select the **Enable stateful session bean failover using memory to memory replication** checkbox. This enables failover for all stateful session

beans in this application. If you want to *disable* the failover, clear this checkbox.

4. Define the Replication settings. You have the choice between two radio buttons (refer to Figure 7-14 on page 381):

- **Use replication settings from EJB container**

If you select this button, any replication settings defined for this application are ignored, that is, you do not overwrite the EJB container settings.

- **Use application replication settings**

Selecting this button overrides the EJB container settings. This button is disabled until you define a replication domain. The selection has a hyperlink to help you configure the replication settings. If no replication domains are configured, the link takes you to a panel where you can create one. If at least one domain is configured, the link takes you to a panel where you can select the replication settings to be used by the application.

Important: When selecting the first radio button (Use replication settings from EJB container), then memory-to-memory replication must be configured at the EJB container level. Otherwise, the settings on this panel are ignored by the EJB container during server startup and the EJB container will log a message indicating that stateful session bean failover is not enabled for this application.

5. Click **OK** and save your changes.

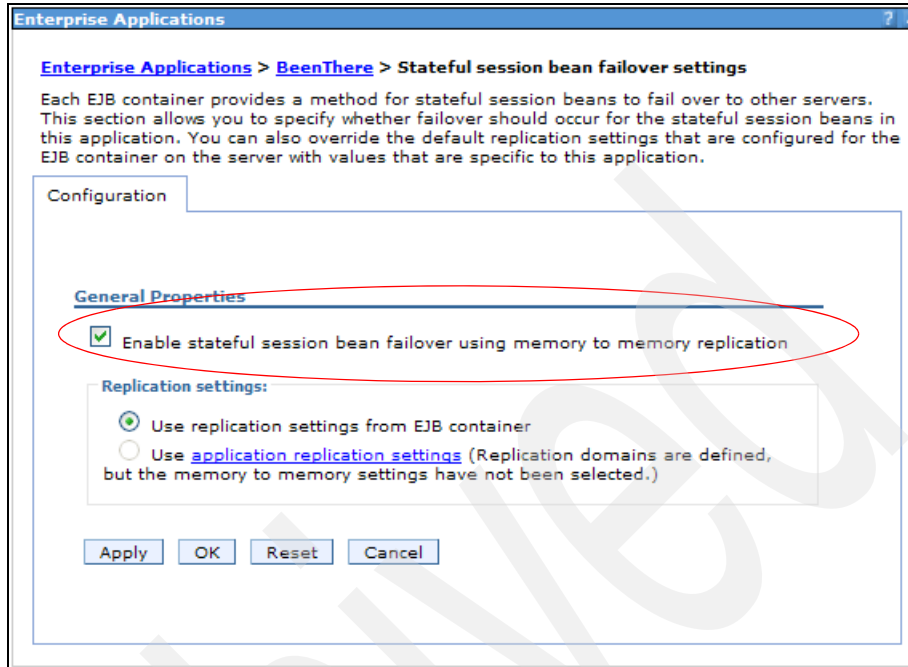


Figure 7-14 Configure stateful session bean failover at application level

Enabling/disabling stateful session bean failover at the EJB module level

1. In the Administrative Console, select **Applications -> Enterprise Applications -> <Application_name>**.
2. Under Related Items, select **EJB Modules**.
3. Select the .jar file you want to work with.
4. Select **Stateful session bean failover settings**.
5. Select **Enable stateful session bean failover using memory to memory replication**.
6. Define the Replication settings as shown in Figure 7-15 on page 382. You have the choice between two radio buttons:
 - **Use application or EJB container replication settings**
If you select this button, any replication settings defined for this EJB module are ignored.

– **Use EJB module replication settings**

Selecting this button overrides the replication settings for the EJB container and application. This button is disabled until a replication domain is defined. The selection has a hyperlink to help configure the replication settings. If no replication domains are configured, the link takes you to a panel where one can be created. If at least one domain is configured, the link takes you to a panel where you can select the replication settings to be used by the EJB container.

Important: If you use the first radio button (Use application or EJB container replication settings), then memory-to-memory replication must be configured at the EJB container level. Otherwise, the settings on this panel are ignored by the EJB container during server startup and the EJB container will log a message indicating that stateful session bean failover is not enabled for this application.

7. Select **OK** and save your changes.



Figure 7-15 Configure stateful session bean failover at EJB module level

Stateful session bean failover best practices

When designing and applying the failover support for stateful session beans on your applications, you should consider the following best practices:

- ▶ If the stateful session bean is still associated with an active transaction or activity session when the failure occurs, the container cannot execute the failover for this stateful session bean. To avoid this possibility, you should write your application to configure stateful session beans to use container managed transactions (CMT) rather than Bean Managed Transactions (BMT).
- ▶ If you desire immediate failover, and your application creates either an HTTP session or a stateful session bean that stores a reference to another stateful session bean, then the administrator must ensure that the HTTP session and stateful session bean are configured to use the same DRS replication domain.
- ▶ Do not use a local and a remote reference to the same stateful session bean. Normally, a stateful session bean instance with a given primary key can only exist on a single server at any given moment in time. Failover may cause the bean to be moved from one server to another, but it never exists on more than one server at a time. However, there are some unlikely scenarios that can result in the same bean instance (same primary key) existing on more than one server concurrently. When that happens, each copy of the bean is unaware of the other and no synchronization occurs between the two instances to ensure they have the same state data. Thus, your application receives unpredictable results.

Important: To avoid this situation, remember that with failover enabled, your application should never use both local (EJBLocalObject) and remote (EJBObject) references to the same stateful session bean instance.

Entity bean failover support

An entity bean represents persistent data. It is common for a client to make a succession of requests targeted at the same entity bean instance. It is also possible for more than one client to independently access the same entity bean instance concurrently. The state of an entity bean must be kept consistent across multiple client requests.

Within a transaction, the WLM ensures that the client is routed to the same server based on the transaction affinity policy. Between transactions, the state of the entity bean can be cached. WebSphere V6 supports Option A, Option B, and Option C caching.

Entity beans can be workload-managed if they are loaded from the database at the start of each transaction. By providing either Option B caching or Option C caching (the default), entity beans can participate in WLM. These two caching

options ensure that the entity bean is always reloaded from the database at the start of each transaction.

See 7.2.3, “Entity beans” on page 345 for a detailed description of the three caching options.

Please note that WebSphere V6 also supports optimistic concurrency control, where the cached data is checked and a collision is detected during the commit stage. Loading data from the database may not be required at transaction start if the application design is in place to stamp cached entity beans.

Note: For more information about WebSphere Application Server behavior regarding optimistic concurrent control, please go to the InfoCenter and search for “concurrency control.”

There are a lot of data and process interactions inside and outside of EJB containers. It is important that one client's failure in an EJB container not impact other clients. It is also important for the underlying data to be kept in a consistent way. According to data availability and local data currency, WLM and failover are not supported where fault isolation and data integrity can be broken.



Part 3

Implementing the solution

Architect

Implementing the sample topology

This chapter provides instructions on how to implement our sample topology. This sample topology is used to demonstrate the WebSphere Application Server V6 scalability, workload management, and high availability features.

We also describe how to install and configure the J2EE applications BeenThere and Trade 6.

The chapter contains the following relevant sections:

- ▶ “Installation summary” on page 392
- ▶ “Installing and configuring WebSphere Edge Components” on page 393
- ▶ “Installing WebSphere and configuring clusters” on page 395
- ▶ “Installing and configuring IBM HTTP Server 6.0” on page 416
- ▶ “Installing and configuring BeenThere” on page 426
- ▶ “Installing and configuring Trade 6” on page 436

Tip: You can go directly to the various sections of this chapter if you do not plan to implement the entire scenario. For example, go to 8.5, “Installing WebSphere and configuring clusters” on page 395 if you do not plan to implement the Caching Proxy and/or Load Balancer but only want to use a cluster to run BeenThere or Trade 6 applications.

8.1 Overview

This section gives you a quick overview of our sample topology. This includes the software needed, a description of our sample topology, and the applications installed in our test environment.

Important: This chapter covers both the AIX platform and the Windows 2000 platform. We describe the implementation of the sample topology on the Windows 2000 platform. However, most steps are exactly the same on both platforms. If there are platform differences, we emphasize this.

8.2 Software products

We are using Windows 2000 and AIX 5.2 in our test environment. The following versions of these operating systems are used:

- ▶ Microsoft Windows 2000 Server with Service Pack 4
- ▶ IBM AIX 5.2 Maintenance Level 2 with 32 bit (unix_up) kernel

The following IBM software is used for the WebSphere implementation:

- ▶ IBM WebSphere Edge Components V6.0
- ▶ IBM JDK 1.4.2
- ▶ IBM HTTP Server 6.0
- ▶ IBM WebSphere Application Server Network Deployment V6
- ▶ IBM DB2 UDB V8.2

More information about the minimum hardware and software requirements for WebSphere and DB2 UDB can be found at:

<http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html>
<http://www.ibm.com/software/data/db2/udb/sysreqs.html>

8.2.1 The sample topology

Our sample topology on the Windows 2000/AIX platform demonstrates the following key features:

- ▶ Use of WebSphere Edge components for caching and Web server load distribution.
- ▶ Web container workload management with WebSphere clusters, both horizontally and vertically (for handling servlet requests).

- ▶ EJB container workload management, both horizontally and vertically.

Figure 8-1 on page 390 illustrates the sample topology:

- ▶ A cluster of two Web servers hosts cluster.itso.ibm.com running the Load Balancer function, anticipated by a Caching Proxy, both from WebSphere Edge Components.
- ▶ A dedicated Deployment Manager machine managing the WebSphere Application Server cell, running IBM WebSphere Application Server Network Deployment V6.
- ▶ Within this cell, we have a WebSphere cluster with various application server processes. IBM WebSphere Application Server Network Deployment V6 is installed on both machines.
- ▶ A dedicated database server running IBM DB2 UDB V8.2.

Important: While this chapter describes how to configure a split JVM topology with the Web containers and EJB containers running in separate application servers, this is done so only for informational and demonstration purposes.

Recall from the note in 3.3, “Strategies for scalability” on page 74 that a negative performance impact will likely result when an application is deployed in this fashion. Better performance, scalability and failover is achieved when the Web components for an application are deployed in the same application server as the EJB components.

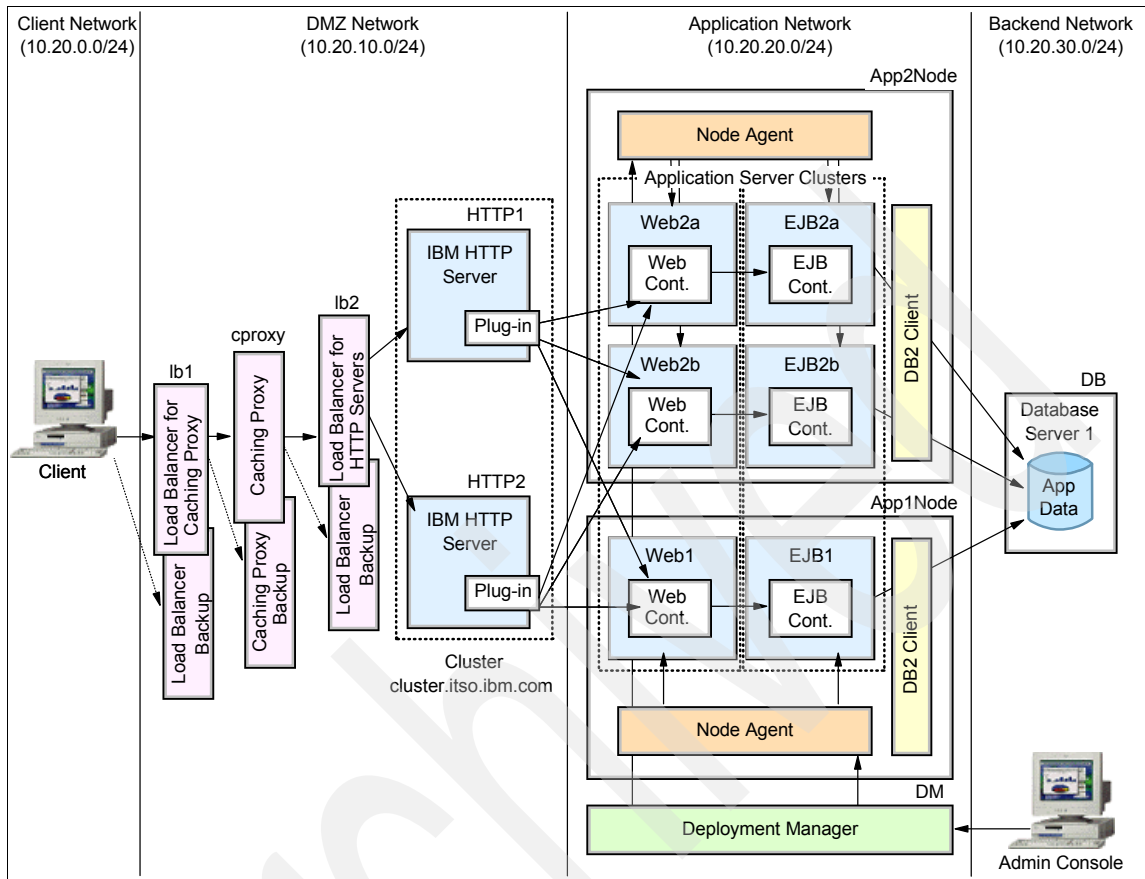


Figure 8-1 The sample topology

Table 8-1 shows the names, functions, locations, and IP addresses of our systems. All machines are part of the same network. However, we choose different IP address ranges to distinguish between the DMZ network, the application network, and the back-end network. In a real production site, these networks would certainly be isolated with firewalls.

Table 8-1 Machines in the sample topology

Name	Network	IP address	Subnet mask	Function
cproxy	DMZ Network	10.20.10.101	255.255.0.0	Caching Proxy
lb	DMZ Network	10.20.10.102	255.255.0.0	Load Balancer
http1	DMZ Network	10.20.10.103	255.255.0.0	Web Server 1

Name	Network	IP address	Subnet mask	Function
http2	DMZ Network	10.20.10.104	255.255.0.0	Web Server 2
dm	Application Network	10.20.20.100	255.255.0.0	Deployment Manager
app1	Application Network	10.20.20.103	255.255.0.0	Application servers system 1
app2	Application Network	10.20.20.104	255.255.0.0	Application servers system 2
db	Back-end Network	10.20.30.100	255.255.0.0	Database server

Table 8-2, shows the properties of our Web server cluster.

Table 8-2 Cluster in the sample topology

Name	Network	IP address	Subnet mask	Function
cluster	DMZ Network	10.20.10.100	255.0.0.0	Web server cluster

Please notice that these tables show only short names for host names. In our sample topology, all host names belong to the `itso.ibm.com` domain. So, for example, the `dm` machine is also known as `dm.itso.ibm.com`.

8.2.2 Applications used in our sample topology

We have used two sample applications in our environment.

BeenThere

BeenThere is a simple, lightweight J2EE application. It is very useful for demonstrating workload management because it shows you which application server responded to a request. For more information about BeenThere, see 8.7, “Installing and configuring BeenThere” on page 426.

Trade 6.0.1

The IBM Trade Performance Benchmark Sample for WebSphere Application Server (called Trade 6 throughout this book) is the fourth generation of the WebSphere end-to-end benchmark and performance sample application.

The Trade benchmark is designed and developed to cover the significantly expanding programming model and performance technologies associated with WebSphere Application Server.

This provides a real-world workload driving WebSphere Application Server V6 implementation of J2EE 1.4 and Web Services, including key WebSphere performance components and features.

Trade 6's new design spans J2EE 1.4, including the EJB 2.1 component architecture, message-driven beans, transactions (one-phase, two-phase commit) and Web services (SOAP, WSDL, UDDI). Trade 6 also highlights key WebSphere performance components such as dynamic caching and WebSphere Edge Components. For more information about Trade 6, see 8.8, "Installing and configuring Trade 6" on page 436.

8.3 Installation summary

Table 8-3 summarizes the installation procedure. These steps need to be carried out *before* proceeding with the rest of this chapter. In general, we do not describe how to install each individual component, but rather we explain how to configure the software that is already installed.

Important: Throughout this chapter, we are using the default application install directories. If you are not using these defaults, make sure to make the proper changes to the instructions provided here.

Tip: Instead of accepting the default installation directory for WebSphere Application Server V6 on the Windows 2000 platform (C:\Program Files\WebSphere\AppServer), we suggest that you install WebSphere in a path that does not have any spaces or other special characters. For example, this could be C:\WebSphere\AppServer.

Table 8-3 Installation summary

Step number	Action	System
1	Install IBM DB2 UDB V8.2 server	db
2	Install IBM DB2 V8.2 client	app1, app2
3	Catalog DB2 node on clients	app1, app2
4	Install IBM WebSphere Application Server Network Deployment V6 (Deployment Manager profile)	dm
5	Install IBM WebSphere Application Server Network Deployment V6 (Managed Node, custom profile)	app1, app2
6	Install IBM HTTP Server 6.0 and WebSphere Application Server V6 plug-in.	http1, http2
7	Install IBM WebSphere Edge Components V6.0	cproxy, lb

There are several resources regarding the installation of these products. You can also find some information in other chapters of this book:

1. IBM DB2 UDB V8.2 client

Check section 8.8, “Installing and configuring Trade 6” on page 436 for some instructions on configuring the system to access a remote DB2 server.

2. IBM WebSphere Application Server Network Deployment V6 profiles

Please refer to *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

You can also check section 8.5, “Installing WebSphere and configuring clusters” on page 395 for a brief description.

3. IBM HTTP Server 6.0 and WebSphere Application Server V6 plug-in

A brief description of the installation of both is found in 8.6, “Installing and configuring IBM HTTP Server 6.0” on page 416.

4. IBM WebSphere Edge Components V6.0

See Chapter 5, “Using IBM WebSphere Edge Components” on page 127 for a description of the installation and configuration of both the Load Balancer and Caching Proxy products.

8.4 Installing and configuring WebSphere Edge Components

We are using both the Caching Proxy and the Load Balancer function of WebSphere Edge Components in our sample topology. Refer to Figure 8-2 on page 394 to see how we have implemented them in our sample topology.

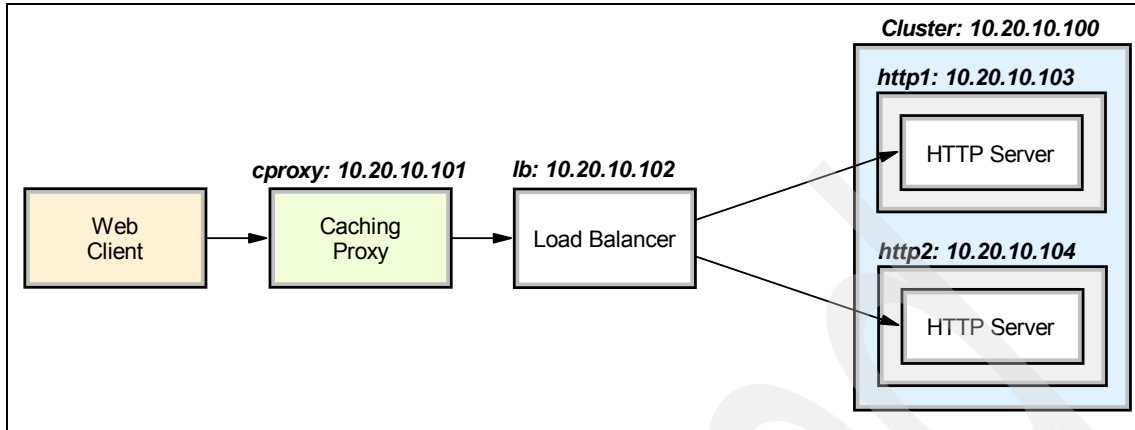


Figure 8-2 Caching Proxy and Load Balancer in the sample topology

8.4.1 Configuring the Caching Proxy

The WebSphere Edge Components V6.0 contains a Caching Proxy feature which allows you to cache often requested documents, thus cutting down the response time.

We have provided step-by-step instructions on how to install and configure the Caching Proxy for our sample topology in 5.6, “Caching Proxy installation” on page 204 and 5.7, “Caching Proxy configuration” on page 212. Therefore, refer to these sections if you wish to set up the Caching Proxy.

8.4.2 Configuring the Load Balancer

The Load Balancer function from the WebSphere Edge Components allows you to implement load balancing for HTTP requests among two or more Web servers. In our sample topology, we defined a cluster named `cluster.itso.ibm.com` with IP address `10.20.10.100`. This is also illustrated in Figure 8-2.

You can find detailed instructions on how to install and configure the Load Balancer in 5.1, “Load Balancer installation” on page 128 and 5.2, “Load Balancer configuration: basic scenario” on page 135.

In short, the configuration steps needed are:

1. The Load Balancer must be configured so that it recognizes the Web server cluster (in our sample topology, the address “`cluster.itso.ibm.com`”) and its members (the Web servers `http1` and `http2`). This configuration step is detailed in 5.2.1, “Configuring the Load Balancer cluster” on page 136.

2. The Web servers (and their machines) must be properly configured so that they can work with the cluster defined by Load Balancer. See 5.2.2, “Configuring the balanced servers” on page 148.

Load Balancer high availability scenario

Refer to 5.3, “Load Balancer: high availability scenario” on page 162 if you also wish to configure a backup Load Balancer for high availability.

8.4.3 Checking the Load Balancer and Caching Proxy configurations

It might be a good idea to check whether everything you configured until now is working. To do so, open a browser and go to the URL:

```
http://cluster.itso.ibm.com/
```

This test shows you whether the Load Balancer can contact its Web server cluster.

The next test proves that you can reach the Web server cluster through the Caching Proxy. Configure the browser to use `cproxy.itso.ibm.com` with port 80 as your connection proxy and try to connect to the same URL again:

```
http://cluster.itso.ibm.com/
```

In both cases, you should see the main page of one of the HTTP servers.

8.5 Installing WebSphere and configuring clusters

This section explains how to configure the WebSphere clusters of the sample topology. In our sample environment, the application is separated into two clusters: one that houses the Web containers, and one that houses the EJB containers. The Web container cluster contains all the servlets and JSP content used to provide presentation services to the application. The EJB cluster houses all the business logic and Enterprise beans that access data services when needed. In addition, we want to provide session failover and thus need to configure persistent session management.

8.5.1 Introduction

Before you can configure the clusters and the session management, you have to make sure that the IBM WebSphere Application Server Network Deployment V6 software is installed on the Deployment Manager machine (dm) and on both application server machines (app1 and app2).

Make sure that you choose to install the Application Server Samples. This includes the BeenThere application which we will install later into our topology. If you prefer not to install the Application Server Samples, then you need to download BeenThere separately from the redbook repository (see Appendix B, “Additional material” on page 1037 for download instructions).

IBM WebSphere Application Server Network Deployment V6 installation has been greatly simplified. The same software is installed on every WebSphere node, upon which one or more profiles are then configured. Thus, after the WebSphere V6 installation, you first configure a Deployment Manager profile on the Deployment Manager machine. As soon as the Deployment Manager is up and running, you can configure the managed node custom profiles on the Application Server nodes.

8.5.2 Deployment Manager installation and profile creation

Starting IBM WebSphere Application Server Network Deployment V6 installation opens the launchpad shown in Figure 8-3.



Figure 8-3 WebSphere installation launchpad

The launchpad allows you to install the WebSphere software (as well as the IBM HTTP server and Web Server plug-ins). Some information is asked for (such as the installation path) before the installation process is completed.

At this point, the user is prompted to run the Profile Creation Wizard, as shown in Figure 8-4. Make sure the checkbox **Launch the Profile creation wizard** is checked and click **Next**.

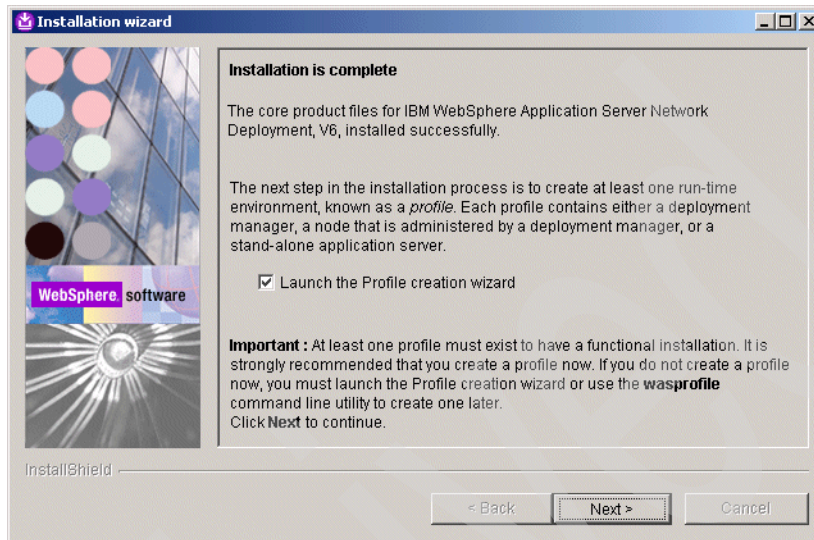


Figure 8-4 Launching the Profile Creation Wizard

Note: After a successful installation, the Profile Creation Wizard can be found in the <WAS_HOME>/bin/ProfileCreator directory.

Choose the correct profile type for the node you are installing. Naturally, on the Deployment Manager machine, you select the Deployment Manager profile and click **Next**, as shown in Figure 8-5 on page 398.

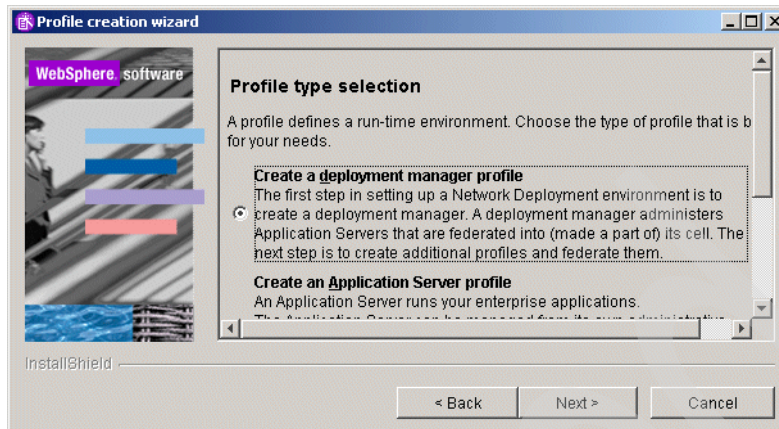


Figure 8-5 Choosing the Deployment Manager profile

The wizard then asks for the profile name, as shown in Figure 8-6. Enter **dm** and click **Next** (dm becomes the default and unique profile on this node).



Figure 8-6 Entering the profile name

In the next window, accept the default profile directory suggested ("`<WAS_HOME>/profiles/dm`") and click **Next**. This brings you to the point where names are asked for the node, host and cell. Fill them in as shown in Figure 8-7 on page 399 and click **Next**.

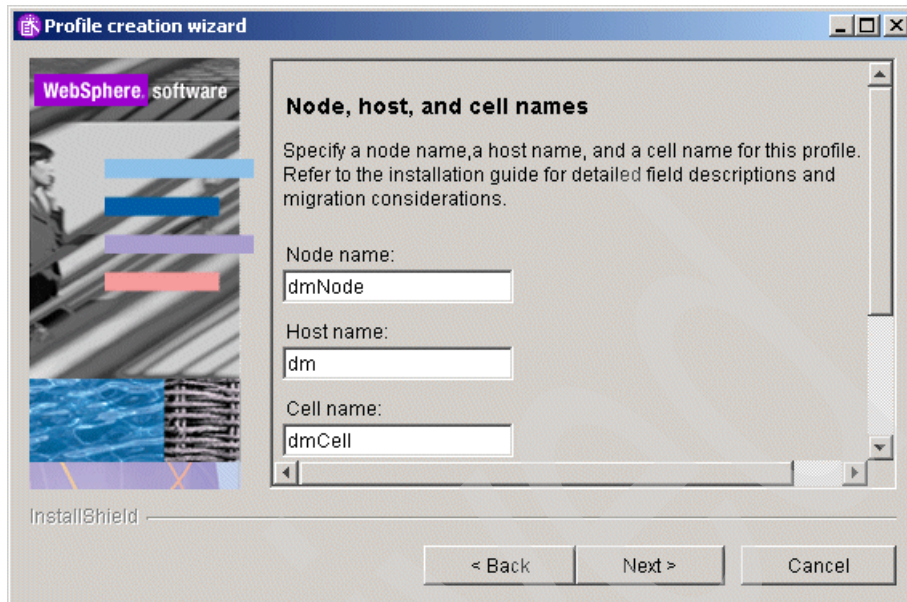


Figure 8-7 Naming cell, host and node

Finally, you are presented with a list of TCP/IP ports that will be used by the Deployment Manager, as detailed in Figure 8-8. Check them for any possible conflicts with previously installed software, then click **Next**.

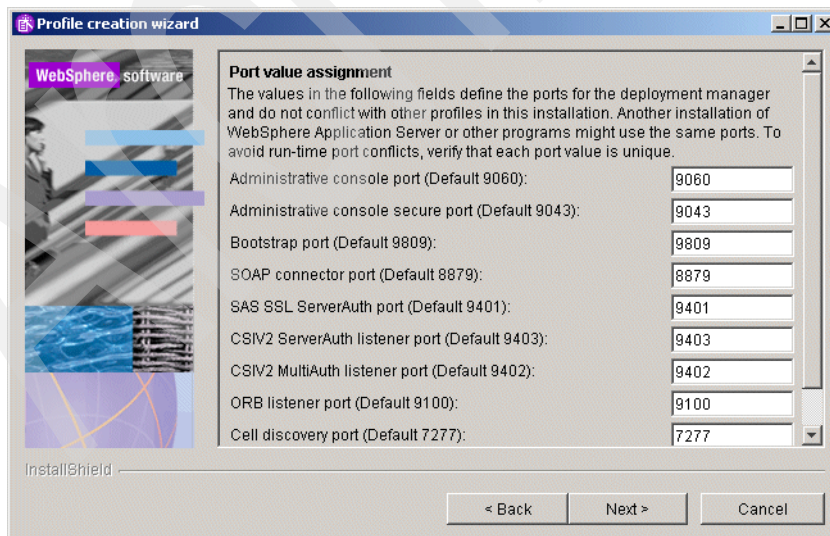


Figure 8-8 Deployment Manager allocated TCP/IP ports

Important: The default Administrative Console HTTP port is now 9060 (no longer 9090).

Windows users can decide in the following window whether the Deployment Manager should be registered as a Windows Service. This is normally recommended, for example to make sure that the Deployment Manager is automatically started after a system restart. If you decide not to register the Deployment Manager as a Windows Service during installation, you can always do it later using the **WASService.exe** command-line utility.

Click **Next** several times until the profile installation begins.

After the installation has ended successfully, use Windows Services or open an OS command prompt and start the Deployment Manager:

```
<WAS_HOME>\bin\startManager.bat (Windows)
```

or

```
<WAS_HOME>/bin/startManager.sh (AIX, Linux, etc.)
```

You are now ready to install and configure the IBM WebSphere Application Server Network Deployment V6 code on the other nodes.

8.5.3 Application server nodes installation (federated nodes)

After configuring the Deployment Manager machine, the next step is to install and configure the app1 and app2 nodes. Both will be federated (managed) nodes of the cell, hosting several application servers.

Install IBM WebSphere Application Server Network Deployment V6 on these nodes in exactly the same way you did on the Deployment Manager machine, except that this time you choose a different profile. In our case, we need a *custom profile*. This profile choice creates an empty node (without servers and applications) that will be federated to an already running Deployment Manager.

1. In the Profile creation wizard, choose **Create a custom profile** (as shown in Figure 8-9 on page 401) and click **Next**.

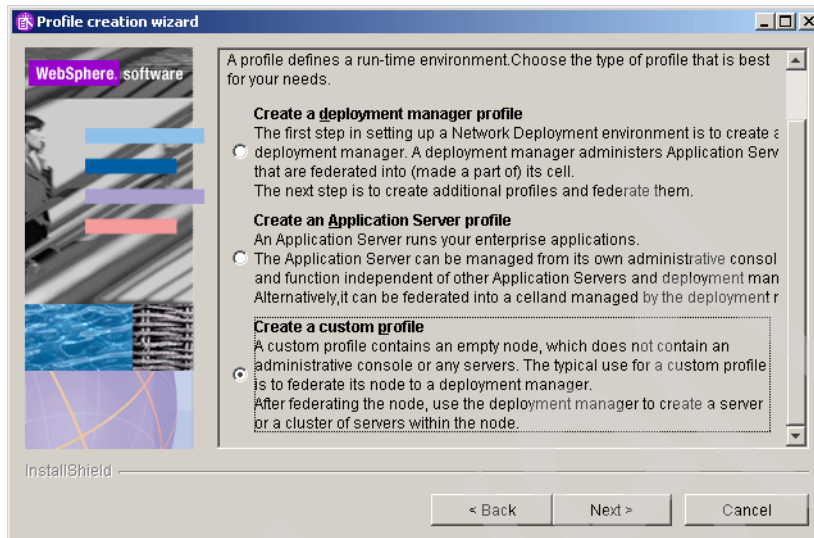


Figure 8-9 Creating a custom profile

2. You must enter the Deployment Manager data, in our case the host name `dm.itso.ibm.com` (or just `dm`) and the default SOAP port `8879` as seen in Figure 8-10.

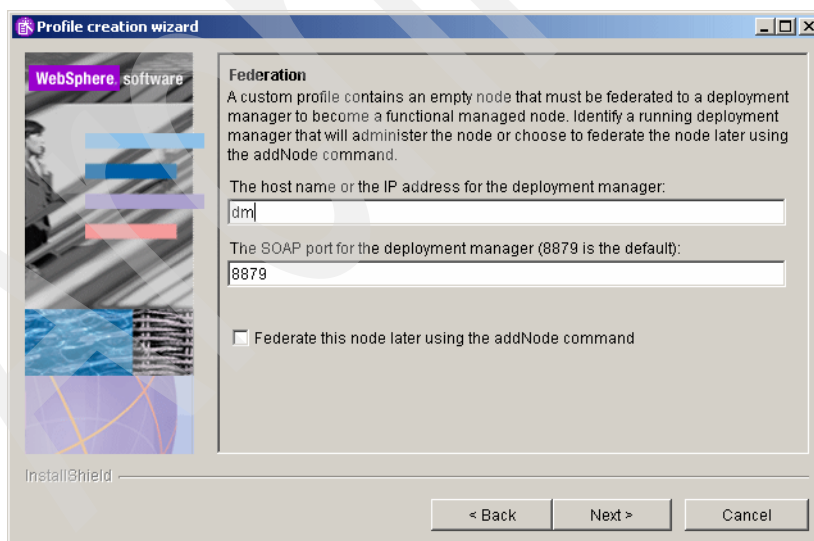


Figure 8-10 Federation data for the managed node

3. Click **Next**. On the next panel, enter the profile name which is either app1 or app2, depending on the machine you are currently installing; then click **Next**.
4. On the next panel, accept the default profile directory and click **Next** once again.
5. In the Node and host names window, enter app1Node or app2Node as the Node name and fill in the proper Hostname (see Figure 8-11). Click **Next**.

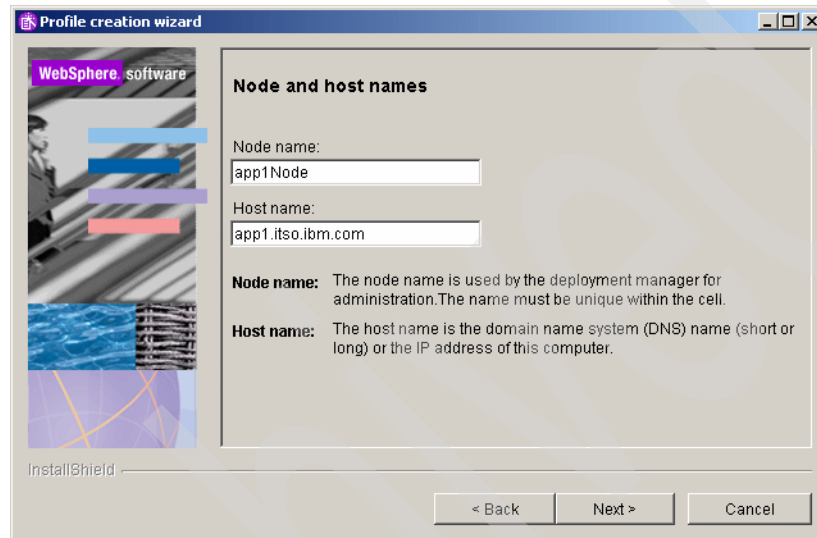


Figure 8-11 Node and host name of the federated node

6. The last panel shows you the suggested TCP/IP ports for this node. Verify them for conflicts (there are usually none since WebSphere checks for used ports) and click **Next**. The profile creation might run for several minutes. Click **Finish**.

8.5.4 Verifying the profiles

The configuration of the cell should now look similar to the one shown in Figure 8-12 on page 403.

To verify this, log on to the WebSphere Administrative Console and select **System administration -> Cell**. Select the **Local Topology** tab and expand **dmCell**. There should be a Deployment Manager node (dmNode) and two application server nodes (app1Node and app2Node).

Please note that so far there are no application servers configured and no applications deployed but a nodeagent is present on both app1Node and

app2Node. So the next step is to create the application server clusters (and their members) in the cell.

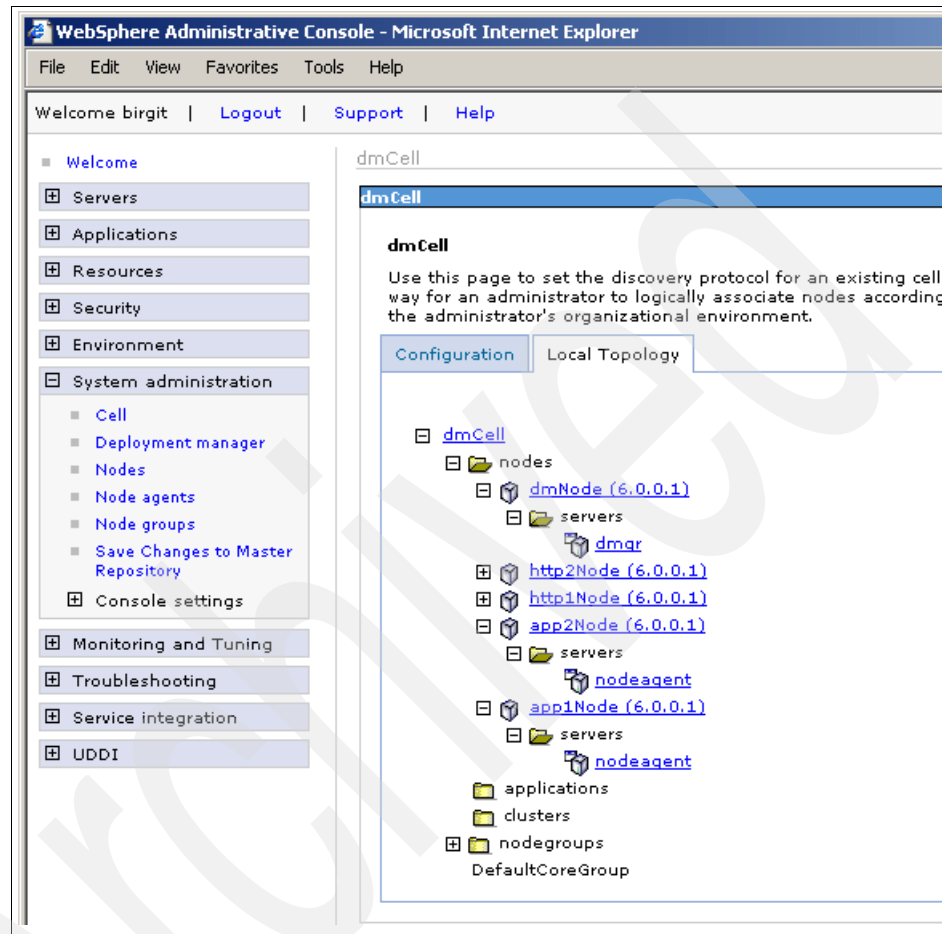


Figure 8-12 The dmCell cell base configuration (no clusters or servers configured)

After verifying this base configuration, you are now ready to start configuring the ITSO sample topology as detailed in Figure 8-1 on page 390.

A high-level description of the procedure for creating the server clusters and its cluster members follows (for details, see 8.5.5, “Creating the Web container cluster” on page 404 and 8.5.6, “Creating the EJB cluster” on page 407):

1. Create the first cluster by using the WebSphere Administrative Console.
Create only the first member of the cluster during cluster creation.

Tip: As a general rule, a cluster's first member becomes the template for the creation of all other members in the cluster. With this in mind, you might wish to finish the cluster creation, configure its first member as desired and then add additional members to the cluster. If you define all cluster members at once during cluster creation, then any particular configuration settings have to be chosen individually for each server afterwards.

2. Configure this first member exactly as you need it.
3. Create the other cluster members.

Note: You can also use a *server template* when creating a cluster's first member. Server templates exist since WebSphere Application Server V5.0 and are used to duplicate an existing application server definition. This definition can be used when creating new application servers with the same configuration.

4. Repeat steps 1 on page 403 to 3 to create the second cluster.
5. Install the enterprise application into the cell. Specify module mappings to place the Web and EJB modules into the appropriate clusters and Web servers.

8.5.5 Creating the Web container cluster

First, we describe how to create the Web cluster, which we decided to call WEBcluster. As mentioned before, this cluster will be used to provide workload balancing and failover for servlets and JSPs. Our cluster consists of three application servers on two different nodes: Web1 on app1Node and Web2a and Web2b on app2Node.

Creating a cluster consists of three steps:

1. Step 1 allows you to enter basic cluster information.
2. Step 2 is to define the cluster members.
3. Step 3 summarizes the information you entered previously.

To create a new cluster:

1. Log on to the WebSphere Administrative Console and select **Servers -> Clusters**. In the right pane, a list of clusters defined within the cell is shown. This list should be empty for a recently installed IBM WebSphere Application Server Network Deployment V6 cell.
2. Click **New** to create your first cluster. This launches the window for Step 1 of the Create a new cluster process as shown in Figure 8-13 on page 405.

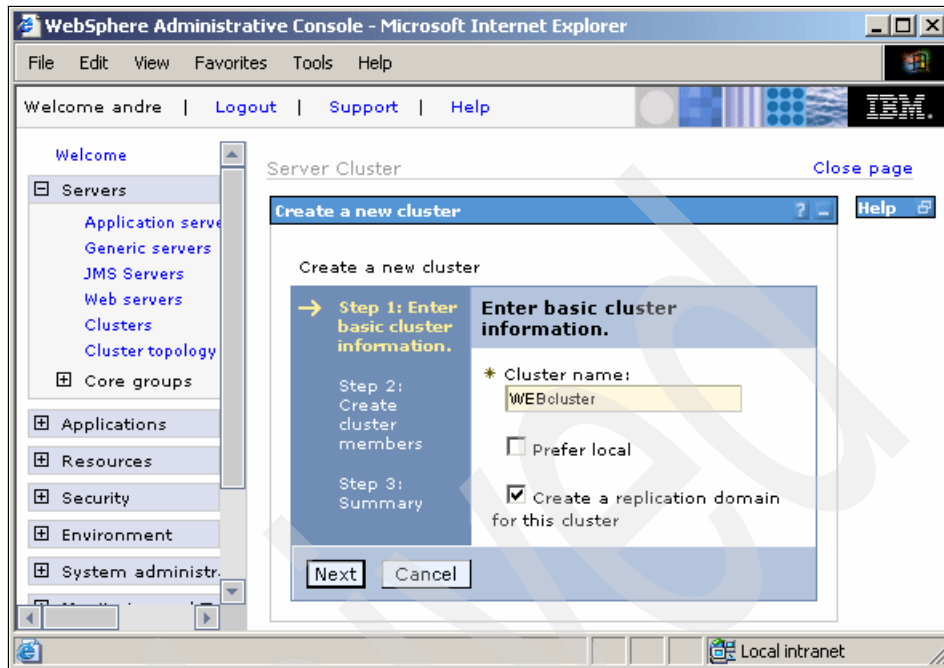


Figure 8-13 Creating a new cluster, Step 1: Enter Basic Cluster Information

3. Enter the basic cluster information:
 - a. Enter WEBcluster for the mandatory Cluster name field.
 - b. Uncheck the **Prefer local** checkbox. Selecting **Prefer local** indicates that EJBs running on the local node should be routed requests to first if the local EJB container is available.

Important: We chose to disable the Prefer Local option in order to demonstrate the workload management features of IBM WebSphere Application Server Network Deployment V6. However, we recommend that this optimization be enabled in a performance-tuned environment.

- c. Check the box **Create a replication domain for this cluster**. This replication domain will be used later when we enable memory-to-memory replication of HTTP session data (for more information about this topic, please see 6.8.7, “Understanding DRS (Data Replication Services)” on page 297). Click **Next** to continue.

This brings up the Step 2 window shown in Figure 8-14 on page 406, which allows you to create new application servers to be added to the cluster.

Create a new cluster

7

Create a new cluster

Step 1: Enter basic cluster information.

→ Step 2: Create cluster members

Step 3: Summary

Create cluster members

Enter information about this new cluster member, and click Apply to add this cluster member to the member list. Use the Edit function to edit the properties of a cluster member that are already included in this list. Use the Delete function to remove a cluster member from this list.

* Member name

Web1

Select node

app1Node(6.0.0.1)

Weight

2

☒ Generate Unique Http Ports

Select template:

☒ Default application server template
Choose a server template from this list:

default

Apply

Figure 8-14 Creating a new cluster, Step 2: Create cluster members

4. To create the three clustered application servers:
 - a. Enter **Web1** as the name for the first new member.
 - b. Select **app1Node** from the Select node pull-down.
 - c. Accept the defaults for all other options and click **Apply** to continue.

Note: We decided to use the default template for all our cluster members rather than changing the configuration of the first application server and then using it as a template for the other members.

The only setting we are changing afterwards is the session management configuration. If you prefer, you can now create only the first member, then go to 8.5.8, “Configure distributed session management” on page 411 and configure the session management for Web1. Then come back to the cluster configuration and add additional members - based on the changed Web1 configuration.

- d. Repeat steps a on page 406 to c on page 406, add a second application server called **Web2a** but this time on **app2Node** (do not click Next until all members have been added).
 - e. Again, using the same procedure, add a third application server called **Web2b** on **app2Node**.
 - f. After adding the three members, click **Next** to continue.
5. Check the summary (see Figure 8-15) and click **Finish** to create the cluster.

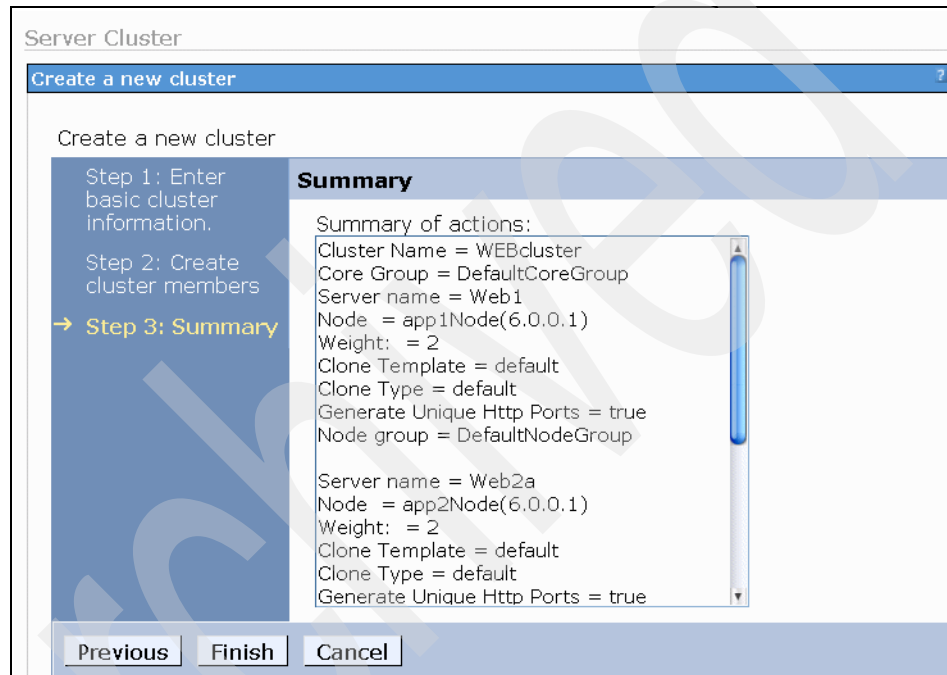


Figure 8-15 Creating a new cluster, Step 3: Summary

6. After completing these steps, the WebSphere Administrative Console warns that changes have been made to the local configuration. Click **Save** in order to save the changes to the master configuration. Select the **Synchronize changes with Nodes** check box to make sure the changes are synchronized to all nodes in the cell. Click **Save** once again.

8.5.6 Creating the EJB cluster

We continue by creating the EJB cluster, which we shall refer to as EJBcluster. This cluster will be used to serve Enterprise beans that access data services when needed, as well as house all the business logic. The EJB cluster also

consists of three application servers on two different nodes: Ejb1 on app1Node and Ejb2a and Ejb2b on app2Node.

1. Log on to the WebSphere Administrative Console (if not logged on already) and select **Servers -> Clusters**. In the right pane, the list of clusters defined for the cell is shown. This time, the WEBcluster should be listed here. Naturally, the second cluster is also created by clicking **New**.
2. Enter the basic cluster information about the Step 1 window of the Create a new cluster process:
 - a. Enter **EJBcluster** as the Cluster name.
 - b. Again, uncheck the **Prefer local** checkbox.
 - c. Accept the default for the other option and click **Next** to launch the Step 2 window.
3. On the Create cluster members window:
 - a. Enter **Ejb1** for the name of the first cluster member.
 - b. Choose **app1Node** from the Select node pull-down.
 - c. Accept the defaults for all other options and click **Apply** to continue.
 - d. Using the same procedure, add another cluster member called **Ejb2a** on **app2Node**.
 - e. Finally, add a third cluster member called **Ejb2b** also on **app2Node**.

Note: When configuring the EJBcluster members, you might notice that you can now select an existing application server as the template for the new cluster members (for example Web1 or Web2a/b). See Figure 8-16. When configuring the WEBcluster you could only select the default template as no other application servers existed in the cell yet.

Server Cluster

Create a new cluster

Create a new cluster

Step 1: Enter basic cluster information.
→ **Step 2: Create cluster members**
Step 3: Summary

Create cluster members

Enter information about this new cluster member, and click Apply to add this cluster member to the member list. Use the Edit function to edit the properties of a cluster member that are already included in this list. Use the Delete function to remove a cluster member from this list.

* Member name

Ejb1

Select node

app1Node(6.0.0.1)

Weight

2

☒ Generate Unique Http Ports

Select template:

☒ Default application server template
Choose a server template from this list:
default

☐ Existing application server
Choose a server from this list:
dmCell/app2Node(6.0.0.1)/Web2a

Apply

Edit

Delete

☒
☐

Select

Application servers

Nodes

Version

Weight

Figure 8-16 Server template selection

After adding these three clustered servers, the WebSphere Administrative Console should look as illustrated in Figure 8-17 on page 410.

Create a new cluster

Step 1: Enter basic cluster information.

→ **Step 2: Create cluster members**

Step 3: Summary

Create cluster members

Enter information about this new cluster member, and click Apply to add this cluster member to the member list. Use the Edit function to edit the properties of a cluster member that are already included in this list. Use the Delete function to remove a cluster member from this list.

* Member name

Select node

app2Node(6.0.0.1)

Weight

2

☒ Generate Unique Http Ports

Apply

Edit Delete

Select	Application servers	Nodes	Version	Weight
<input type="checkbox"/>	Ejb1	app1Node	6.0.0.1	2
<input type="checkbox"/>	Ejb2a	app2Node	6.0.0.1	2
<input type="checkbox"/>	Ejb2b	app2Node	6.0.0.1	2

Previous Next Cancel

Figure 8-17 Create EJBcluster with 3 members

- Click **Next** to continue, then check the summary and click **Finish** to create the EJBcluster.
- After completing these steps, the console once again warns that changes have been made to the local configuration. Click **Save** in order to save the changes to the master configuration and do not forget to synchronize with the cell's nodes.

8.5.7 Verifying the cluster topology

Now it is time to verify the cluster topology. Using the WebSphere Administrative Console, click **Servers -> Cluster topology**. After expanding all items in the right pane, the cluster topology looks as shown in Figure 8-18 on page 411.

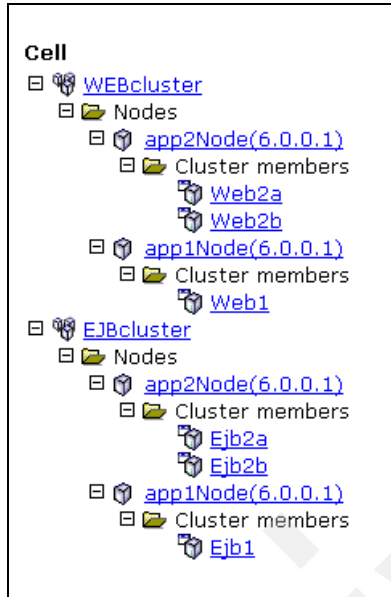


Figure 8-18 The cluster topology

The next step is to configure the WEBcluster servers to support distributed HTTP sessions.

8.5.8 Configure distributed session management

As you know, WebSphere Application Server V6 supports distributed session management in two ways:

- ▶ Database session persistence, where sessions are stored in a defined database.
- ▶ Memory-to-memory session replication, where sessions are replicated to one or more application server instances.

For more information about distributed session management, see “Session persistence considerations” on page 78 and refer to Chapter 6, “Plug-in workload management and failover” on page 227, especially to “Session management configuration” on page 287.

For our sample topology, we decided to use memory-to-memory replication. The following section shows you how to configure this function.

When creating our WEBcluster, we also created a replication domain. Thus, the first thing to do is to verify that the replication domain has indeed been created:

1. Log on to the WebSphere Administrative Console and select **Environment -> Replication domains**. You should see the replication domain WEBcluster in the list.
2. Click the **WEBcluster** replication domain. The replication domain configuration properties appear as shown in Figure 8-19.

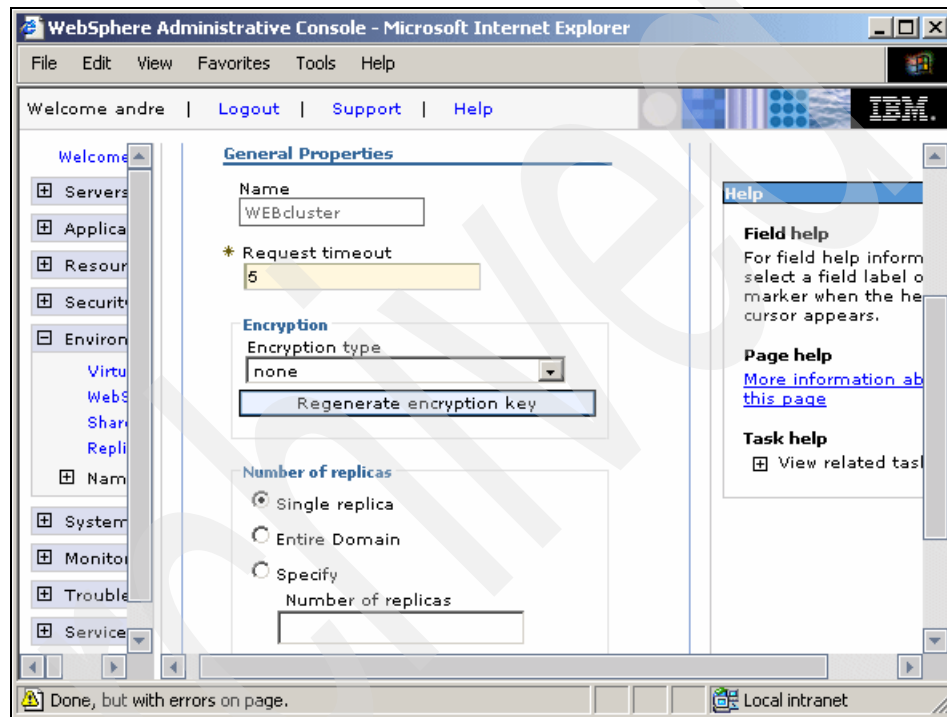


Figure 8-19 Replication domain configuration

The default setting is ok for us (a single replica for each HTTP session), so nothing needs to be changed.

Note: If you did not create the replication domain along with the cluster, you can create a new one now.

The following instructions need to be repeated for each application server in the WEBcluster (Web1, Web2a, and Web2b). Please remember that normally it is recommended that you create a single member first, configure it further and only then create the additional cluster members so that these will have the same

settings. In our case, this approach would have avoided that we have to change the three different members individually now.

1. Using the WebSphere Administrative Console, select **Servers -> Application servers** and select the **Web1** application server.
2. Expand **Web Container Settings** and select **Session management**.
3. On the next window, in the Additional Properties section, select **Distributed environment settings**. The panel shown in Figure 8-20 is displayed.

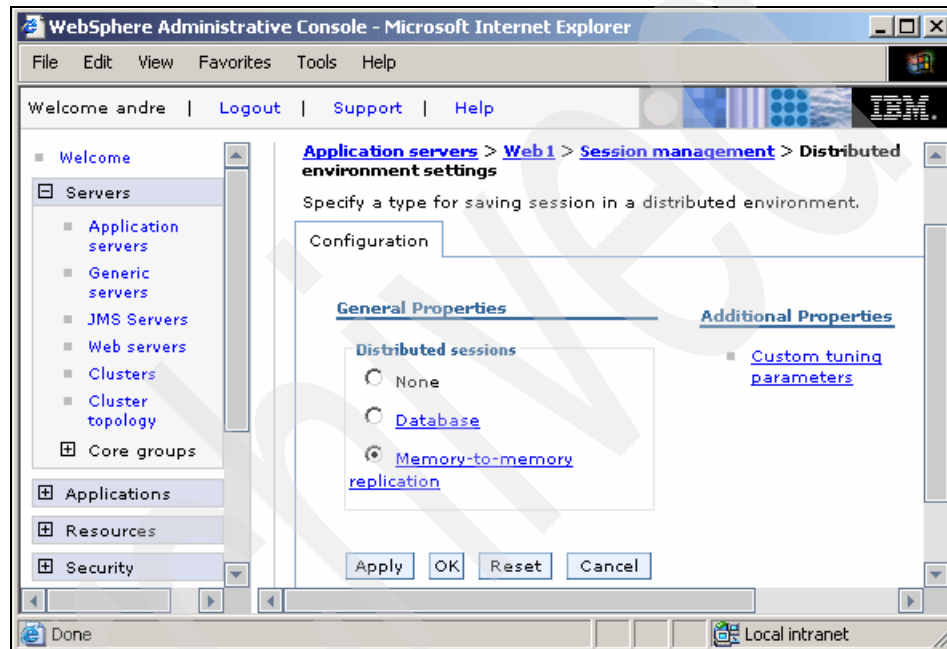


Figure 8-20 Select Memory-to-memory replication

4. Check the **Memory-to-memory replication** radio button. This automatically opens the next panel where additional settings can be configured. See Figure 8-21 on page 414.

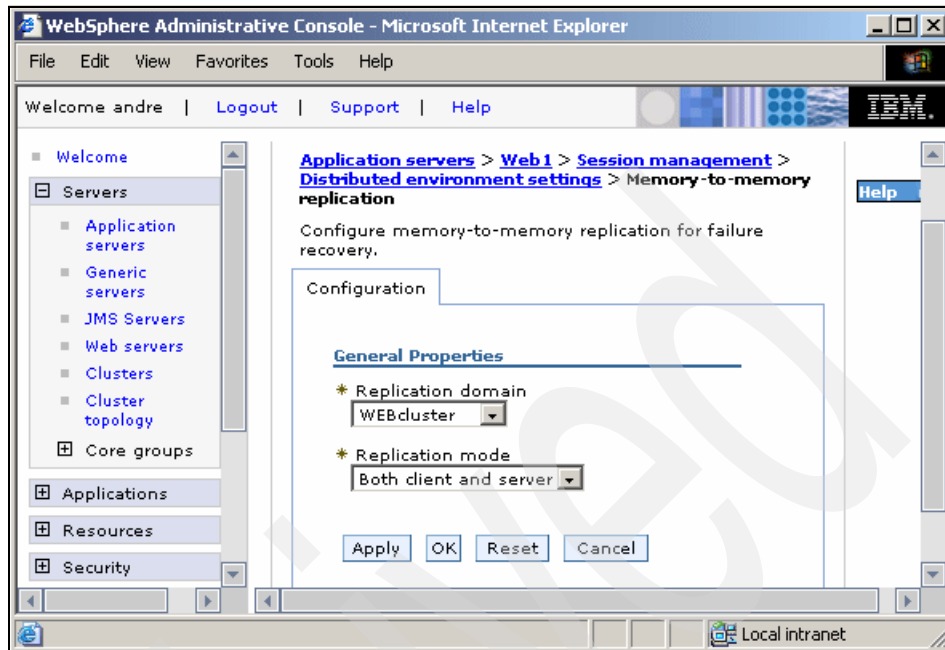


Figure 8-21 Configuring memory-to-memory session replication

5. Select **WEBcluster** as the Replication domain and ensure the Replication mode is set to **Both client and server**. Click **Apply**.

Important: In a cluster whose members have enabled memory-to-memory session replication, sessions are replicated to other servers in the cluster that participate in the same replication domain. There can be as many session replicas as you want (see Figure 8-19 on page 412), the default being a single replica for each session.

When the Single replica option is chosen, one single server in the replication domain is selected during session creation, and all updates to the session are only replicated to that single server.

If the Entire Domain option is chosen each application server will replicate session data to all other replication domain members - this should be regarded as the most robust choice for memory-to-memory replication. However, performance could be affected by this (if you have many application servers) and you should consider this choice carefully. Using fewer replicas (you can specify the number of replicas in the replication domain) or database persistence might be the better choice for your production environment.

6. Click the **Distributed Environment Settings** link at the top of the window and select **Custom tuning parameters** (under Additional Properties).
7. Select **Low (optimize for failover)** as the Tuning level and click **Apply**. See Figure 8-22. This ensures an update of all session data of a particular session before the servlet engine finishes its request.

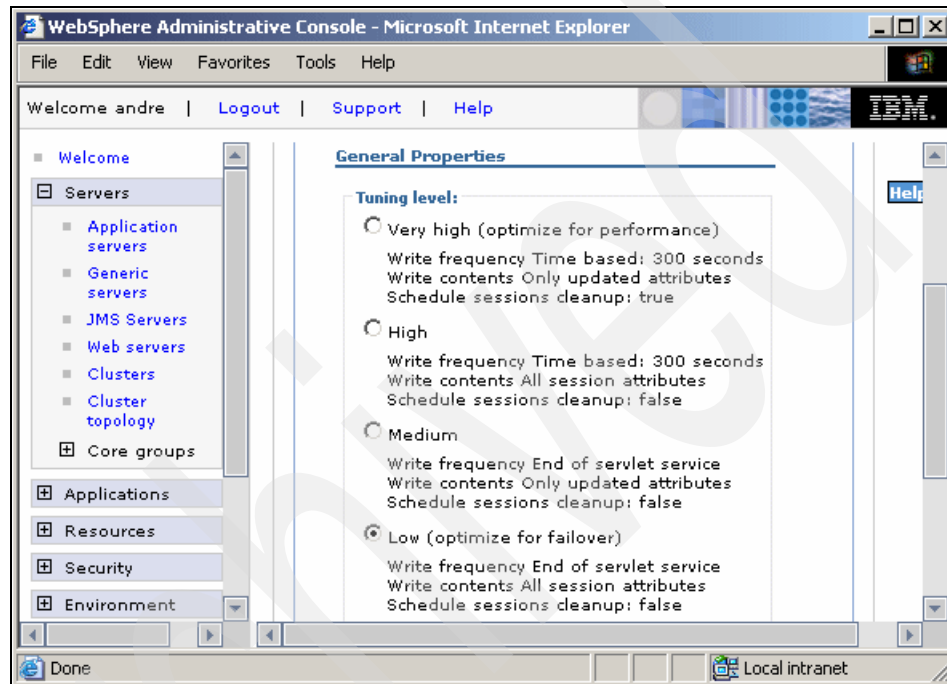


Figure 8-22 Tune the distributed session management

8. Click **Save** to save the changes to the master configuration.
9. Repeat step 1 on page 413 to step 8 for all other cluster members.

Note: In case you did not create all cluster members at cluster creation time (8.5.5, “Creating the Web container cluster” on page 404) but chose to use the “Configure one member -> change it -> add additional members” approach, you must now go back to your cluster configuration and add Web2a and Web2b using Web1 as the server template.

8.5.9 Starting the clusters

It is now time to start the newly created clusters via **Servers -> Clusters**. Select both clusters and click the **Start** button. Use the **Refresh** icon to verify the status.

Tip: You can click the  icon to select all objects in a view (here all clusters).

8.6 Installing and configuring IBM HTTP Server 6.0

In this section we cover the IBM HTTP Server 6.0 and WebSphere plug-in installation and configuration.

8.6.1 IBM HTTP Server 6.0 installation

IBM HTTP Server 6.0 installation is quite easy. It can be started from the same launchpad as is used to run WebSphere V6 installations, as shown in Figure 8-23:

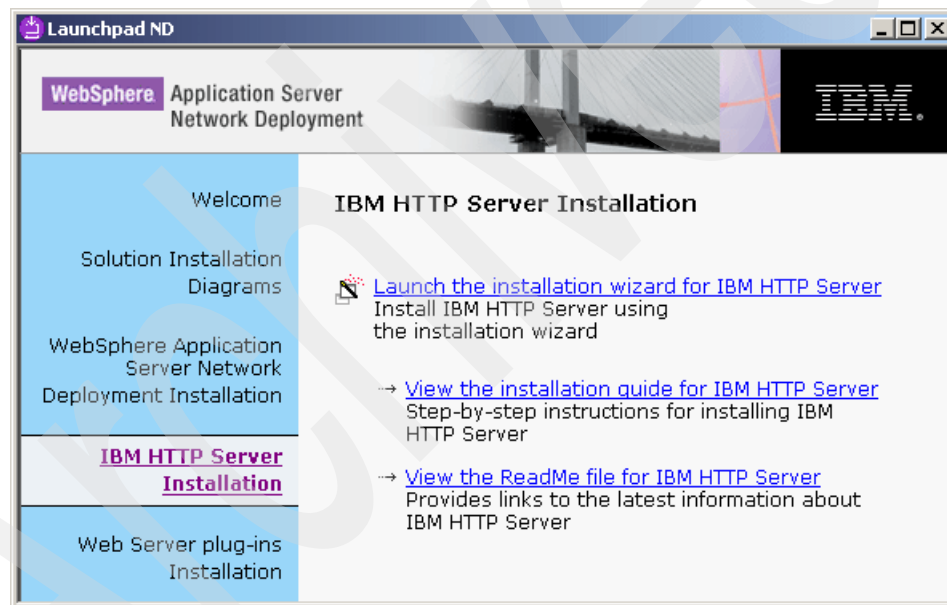


Figure 8-23 Launchpad: IBM HTTP Server installation

You should write down the installation path and ports that you enter (or accept) during IBM HTTP Server 6.0 installation. These are needed later when you configure Web servers in the WebSphere cell. Please remember that IBM HTTP Server 6.0 consists of a Web server (which defaults to port 80) and an Administration server (defaults to port 8008).

Tip: When dealing with distributed and portable software it is always a good idea to avoid directory names with spaces. Instead of accepting the default directory for IHS (C:\Program Files\IBM HTTP Server), you may wish install it in C:\IHS.

IBM HTTP Server Administration server login/password

After IBM HTTP Server 6.0 installation you should set the administrative login user and password for the Administration server. These are required later.

To set a new administrative userid (and its password) you must use the **htpasswd** command. For a Windows machine use this command:

```
C:\IHS\bin\htpasswd.exe -b -c C:\IHS\conf\admin.passwd <userid> <password>
```

And for an AIX machine:

```
/usr/IHS/bin/htpasswd -b -c /usr/IHS/conf/admin.passwd <userid> <password>
```

8.6.2 WebSphere plug-in installation

From the last step of the IBM HTTP Server 6.0 installation you can launch the *WebSphere Application Server - Plugin* installation (you can call it also separately using the launchpad). This is a quick and easy installation that can be resumed to a few simple steps:

1. **Web server choice:** You must select which Web server you are using as the plug-in differs for the various Web servers (see Figure 8-24 on page 418).

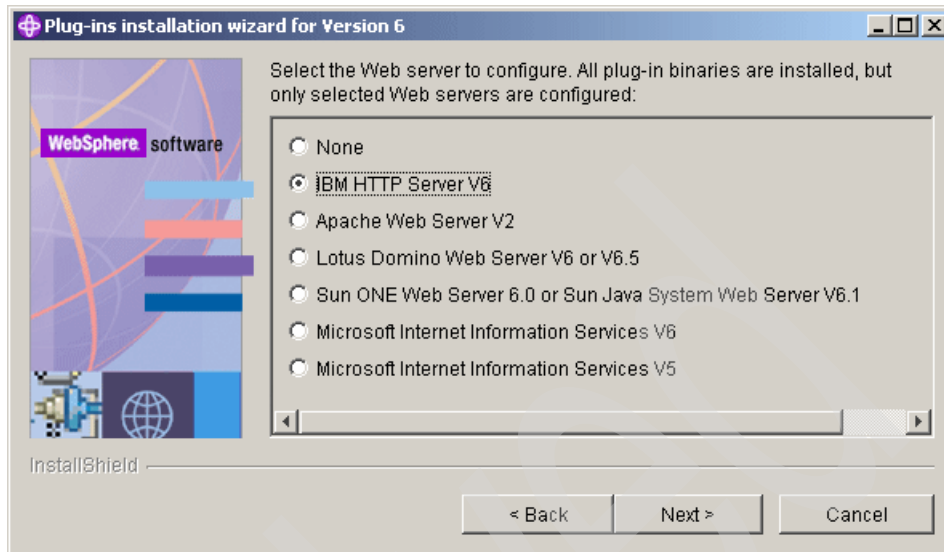


Figure 8-24 Web server choice for plug-in installation

2. **Scenario choice:** You must select whether the Web server runs on a WebSphere node (in other words, a machine that runs a Node Agent) or not. The options presented are local (WebSphere node) or remote, as shown in Figure 8-25 on page 419.

In our sample topology, both Web servers are remote.

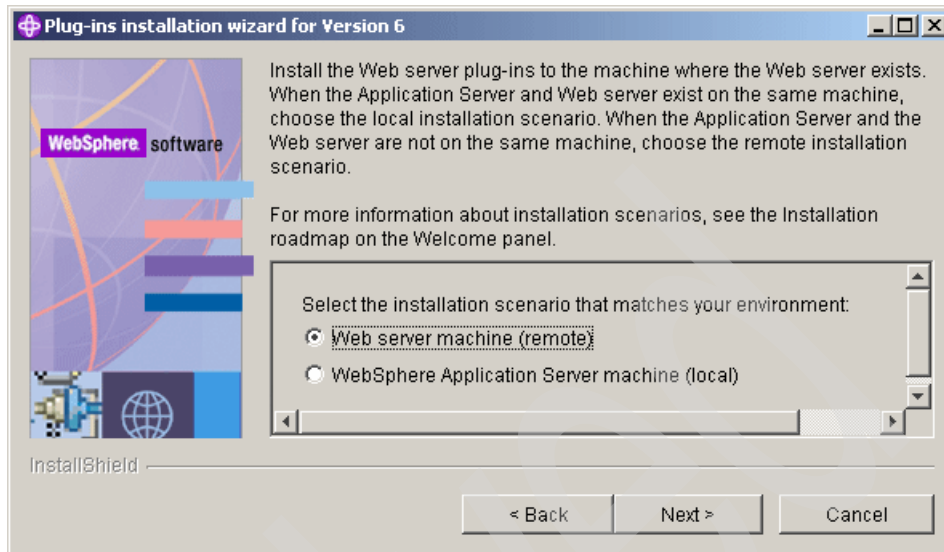


Figure 8-25 Scenario choice for plug-in installation

3. **Installation path:** Enter the desired installation path for the WebSphere plug-in.

Tip: The WebSphere plug-in can be installed in C:\WebSphere\Plugins instead of the default path C:\Program Files\IBM\WebSphere\Plugins to avoid using spaces.

4. **Path to configuration files/Web server port:** Enter the location of the IBM HTTP Server 6.0 configuration file (this must be <IHS_HOME>/conf/httpd.conf) and the HTTP port (most likely port 80).
5. **Web server definition name:** Enter the name you plan to use when defining this Web server in the cell (we used the names http1 and http2 for the Web servers in our sample topology).
6. **Plug-in configuration file location:** Accept the default path suggested.
7. **Application Server location:** Enter the name of one of your application servers.

This application server name will be used in priming the default plug-in configuration file for the Web server. This gives you the best possible “out-of-the-box” experience. Immediately after the installation the plug-in starts routing the default applications. Additional custom applications can then be added later but require the regeneration and propagation of the plug-in.

You must install IBM HTTP Server and the WebSphere plug-in on both http1 and http2 machines, according to our sample topology.

All further Web server configuration is done through the WebSphere Administrative Console (Web server and plug-in settings). The IBM HTTP Server Administration server does not provide a separate HTML Administrative Console.

To start the IBM HTTP Server and its Administration server on a Windows machine you can use the Control Panel/Services utility.

Note: In case you cannot start the HTTP Server as a Windows Service, it may be helpful to run <IHS_HOME>\bin\Apache instead, which may provide additional information if an error occurs.

To start the IBM HTTP Server on AIX use the following command (we assume the install path is /usr/IHS):

```
/usr/IHS/bin/apachectl start
```

And to start the Administration server:

```
/usr/IHS/bin/adminctl start
```

After the WebSphere plug-in installation completed, you can verify the IBM HTTP Server configuration file (<IHS_HOME>/conf/httpd.conf) for the plug-in module load command (these lines are normally at the very end of the file):

```
LoadModule was_ap20_module "C:\WebSphere\Plugins\bin\mod_was_ap20_http.dll"  
WebSpherePluginConfig "C:\WebSphere\Plugins\config\http1\plugin-cfg.xml"
```

The first line loads the WebSphere plug-in (it is physically a dynamic library); the second line specifies the plug-in configuration file location (please notice that the Web server name (here http1) is part of this path).

8.6.3 Configuring Web servers in the cell

Web servers (whether managed or not) are now conceptually a part of the cell topology. Since WebSphere Application Server V6, installed applications are to be targeted to Web servers as well as to application servers (or clusters), as discussed in 6.4, “Web server topologies” on page 244, so the Web server definition is quite important.

Managed or unmanaged nodes

Before adding an already installed Web server to the cell you must make sure that its node is registered in the cell. If this Web server runs on a machine that

also runs a Node Agent process then it is a managed Web server (running on a managed node). If this Web server runs on an isolated machine (without a Node Agent), then we are talking about an unmanaged Web server (running on an unmanaged node). More on this subject can be found in 6.4, “Web server topologies” on page 244.

Unmanaged nodes must be added to the cell using the WebSphere Administrative Console prior to configuring its Web servers. Managed nodes are already defined (or else there would be no Node Agent on them). Both Web servers in the sample topology (http1 and http2) are unmanaged, running on the unmanaged nodes http1Node and http2Node.

Therefore, the steps to configure the Web servers in our topology are:

1. Configure the unmanaged nodes
2. Create the Web server definitions

Configure unmanaged node(s)

To create the unmanaged nodes for the sample topology, you must follow these steps:

1. Select **System administration -> Nodes -> Add Node**. This launches the panel shown in Figure 8-26 on page 422:

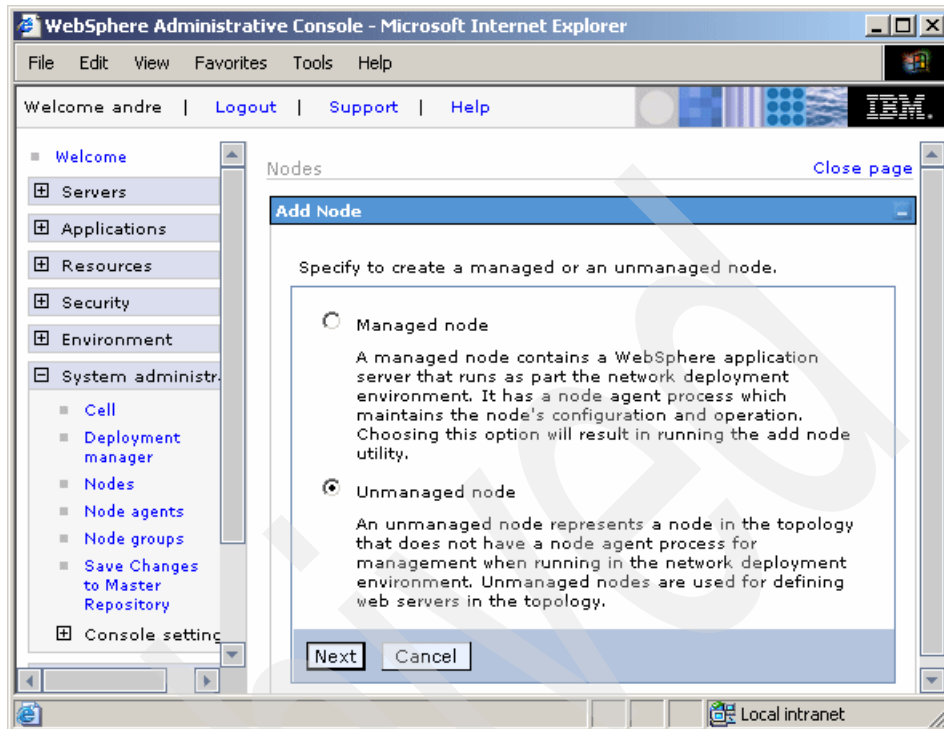


Figure 8-26 Configuring an unmanaged node

2. Select **Unmanaged node** and click **Next**.
3. In the next window, enter the new node name (http1Node), its host name (in our case, http1.itso.ibm.com, or just http1) and the OS platform it runs on, as seen in Figure 8-27 on page 423:

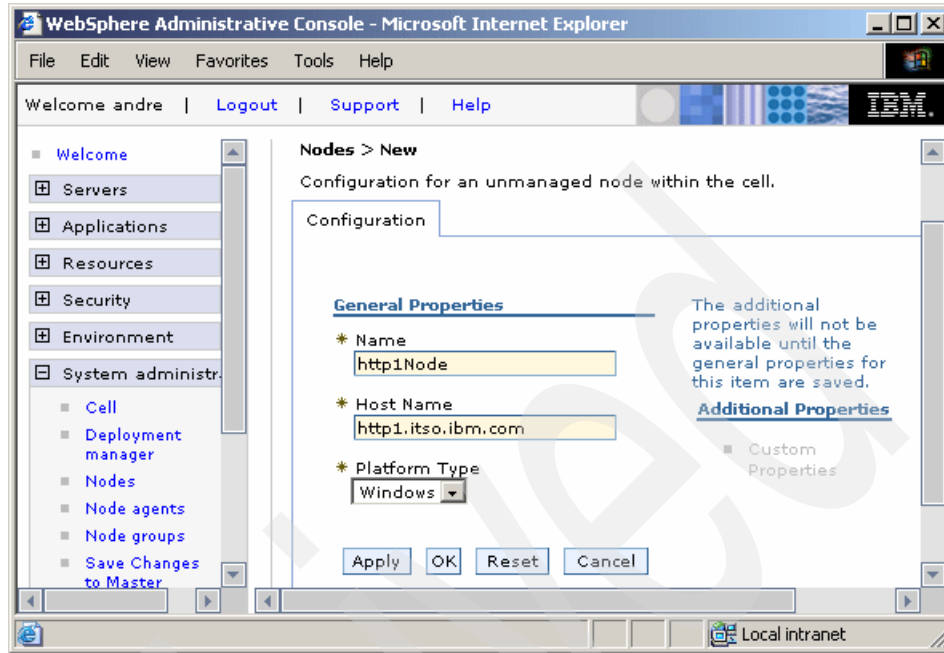


Figure 8-27 Entering unmanaged node information

4. Click **Ok**.
5. Repeat steps 1 on page 421 to 4 to configure the http2Node.
6. Save and synchronize the changes.

Create Web server definitions

To add a Web server to the cell topology, follow these steps:

1. Select **Servers -> Web servers** and click **New**. This opens the panel shown in Figure 8-28 on page 424 (Step 1 - Select a node).

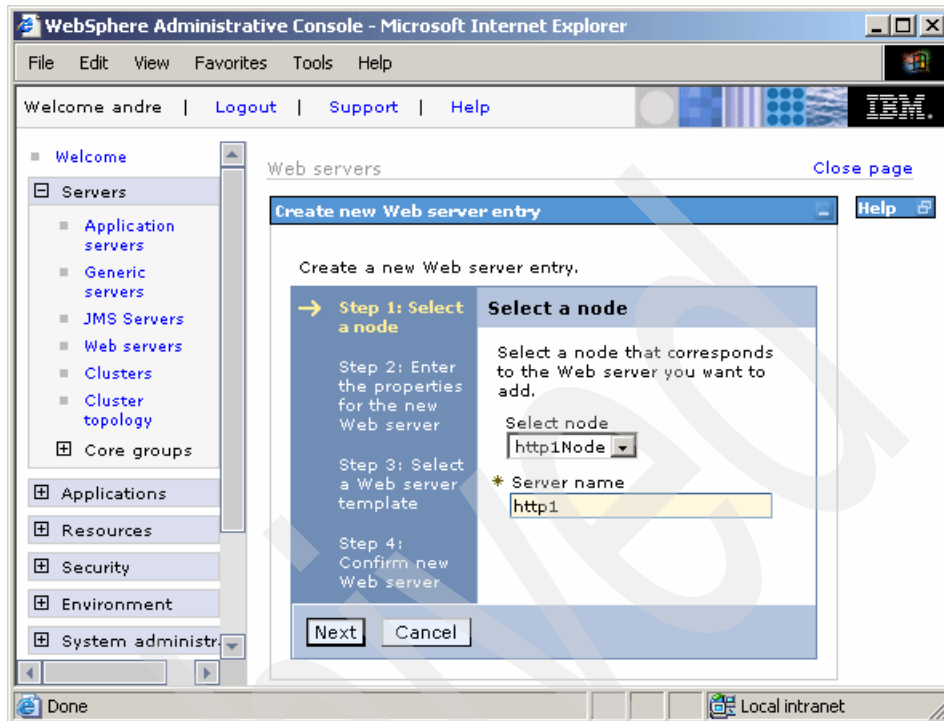


Figure 8-28 Adding a Web server to the cell topology

2. Select the appropriate node (**http1Node**) and enter the server name `http1`. The server name is a WebSphere object name, not a host name - the host name is implied when you pick the node. Click **Next**.
3. On the next window (Figure 8-29 on page 425, Step 2) you must enter several Web server properties, like its type (IHS, Apache, IIS etc.), port, installation path and plug-in installation path. Please notice that the Service name field is important on Windows machines for starting and stopping the Web server from the WebSphere Administrative Console (on a managed node or when using IBM HTTP Server). Fill in all relevant fields and click **Next**.

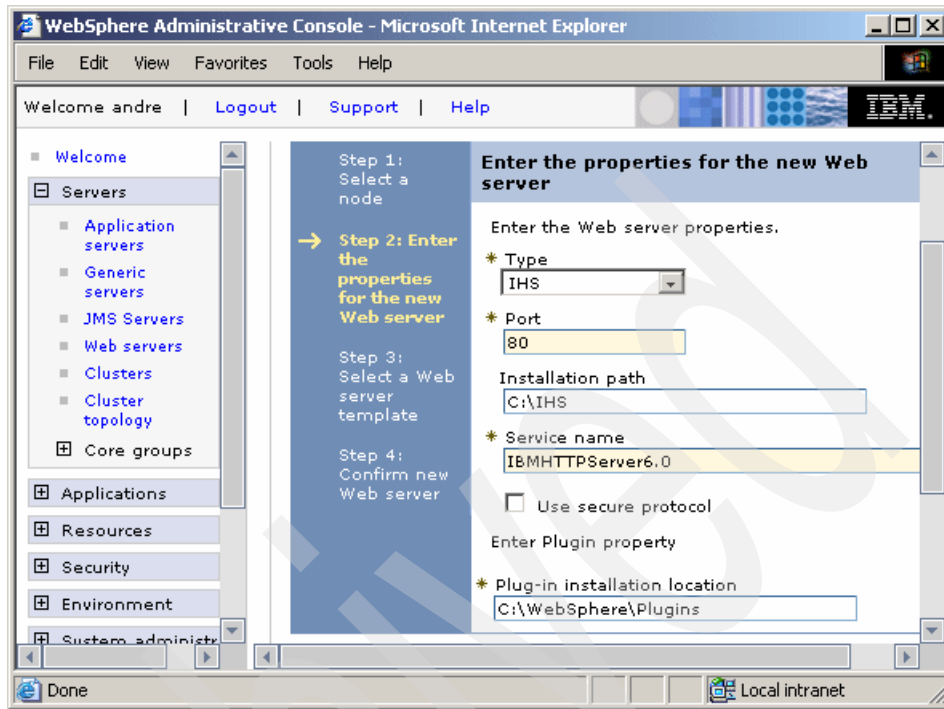


Figure 8-29 Entering Web server properties

4. On the next window (still Step 2) you have to enter the administration properties, such as User ID and Password, for the Web server. Click **Next**.

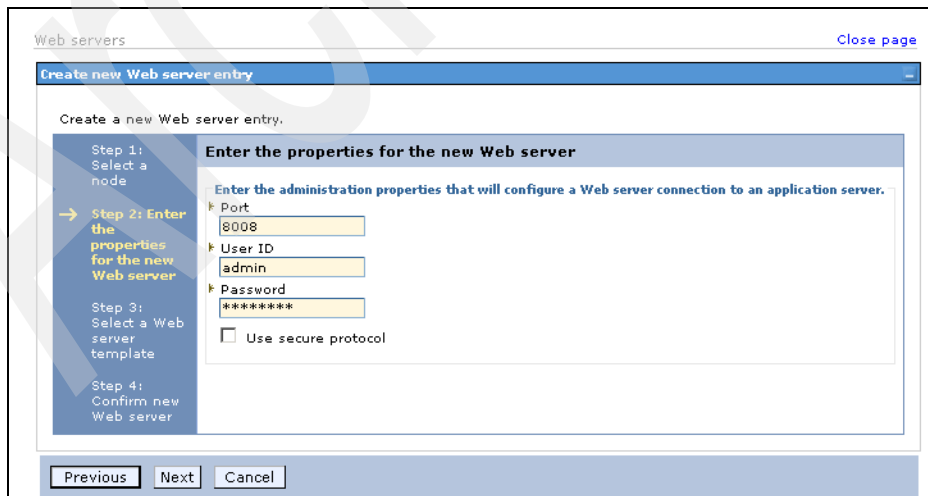


Figure 8-30 Administration properties for the Web server

Important: User ID and Password are case-sensitive.

5. On the next window (Step 3) you simply select **The IHS WebServer Template** and click **Next**.
6. Finally, on the last window (Step 4) verify the summary and click **Finish**.
7. Save your changes and synchronize with the nodes.
8. Repeat steps 1 on page 423 to 7 to create the http2 Web server definition on node http2Node.

8.6.4 Testing Web server configurations

You should now be able to start and stop managed Web servers and IBM HTTP Server unmanaged servers from the **Servers -> Web servers** pane. Please remember that IBM HTTP Server unmanaged servers are a special case that offer the same administrative features as managed servers.

Now that our Load Balancer, Caching Proxy, Web servers, and application servers are up and running, we can install our two sample applications (BeenThere and Trade 6) into the sample topology.

8.7 Installing and configuring BeenThere

The BeenThere application can be used to actually see workload management (WLM) of HTTP requests and of Enterprise JavaBean (EJB) requests. BeenThere consists of the BeenThere servlet, which is used to demonstrate to which application server in a cluster an HTTP request was dispatched and of the BeenThere stateless session EJB used to demonstrate to which application server in a cluster an EJB request was dispatched.

BeenThere is one of the Application Server Samples that can optionally be installed with IBM WebSphere Application Server Network Deployment V6. If you installed the application samples, then you find BeenThere in the following directory:

```
<WAS_ND_install_root>/samples/lib/BeenThere
```

Documentation for BeenThere is found at

```
<WAS_ND_install_root>/samples/readme_BeenThere.html
```

In addition, we have placed the BeenThere code into the additional materials repository of this redbook. Refer to Appendix B, “Additional material” on page 1037 for instructions on how to obtain it.

The BeenThere Enterprise Application Resource (.ear) file is all we need.

8.7.1 BeenThere installation summary

The following actions have to be performed in order to set up and install BeenThere:

- ▶ Install the application
- ▶ Regenerate the Web server plug-in (plugin-cfg.xml)
- ▶ Check server's configuration
- ▶ Restart the servers

8.7.2 Install BeenThere

To install and deploy the BeenThere Enterprise Application Archive file, follow these steps:

1. Log on to the WebSphere Administrative Console and click **Applications -> Install New Application**.
2. The Preparing for the application installation window appears as shown in Figure 8-31 on page 428.
 - a. Here you can specify the EAR/WAR/JAR module to upload and install.

By default, the BeenThere.ear file is found in
<WAS_ND_install_root>/samples/lib/BeenThere.

Click the **Browse** button to specify the location of the file, select it and click **Next** to continue.

Note: The .ear file can reside on any node in the cell or on the local workstation running the WebSphere Administrative Console. Depending on where you stored it, use the Browse button next to the Local file system or Remote file system selection.

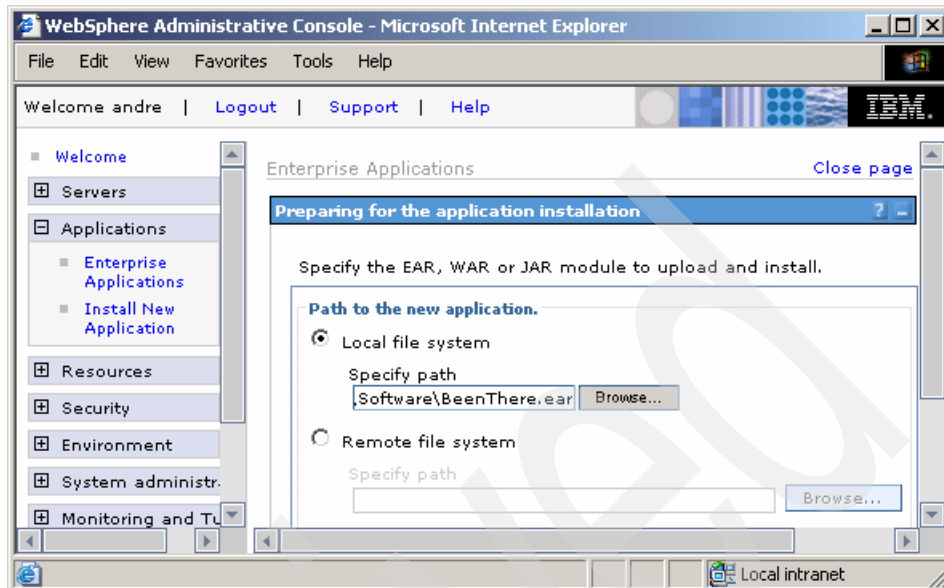


Figure 8-31 Specify the location of the BeenThere ear file

- b. The next window allows you to define mappings and bindings. We do not need to change anything here, so click **Next** to accept all defaults.
 - c. The next panel shows warnings regarding the policy file included in the EAR. Since we have not enabled Java 2 security in our cell you can ignore it and click **Continue**.
3. The upcoming window shows the first of several steps in the Install New Application process.
 - a. Select to **Deploy enterprise beans** during the installation.
 - b. Make sure the Application name **BeenThere** is correct as shown in Figure 8-32 on page 429 and click **Next**.

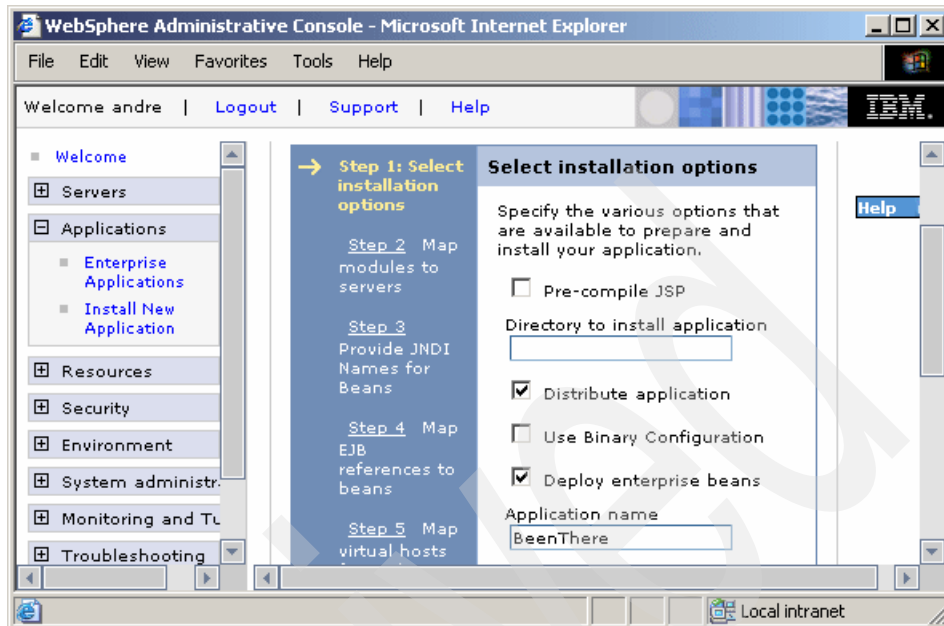


Figure 8-32 Install New Application - Installation options

- c. On the next step (Map modules to servers) you must map the application's modules to the proper servers. Remember that we want to run Web modules on the **WEBcluster** and EJBs on the **EJBcluster**. In addition, the Web modules must also be mapped to the Web servers. You can map a module by checking it, selecting the application or Web server(s) from the Clusters and Servers list and clicking **Apply**. For our scenario:
 - i. Select the **BeenThere WAR** Module, then the **WEBcluster** and the Web servers **http1** and **http2** and click **Apply**.
 - ii. Select the **BeenThere EJB** Module and the **EJBcluster** and click **Apply**.

The final mapping can be seen in Figure 8-33 on page 430. Click **Next**.

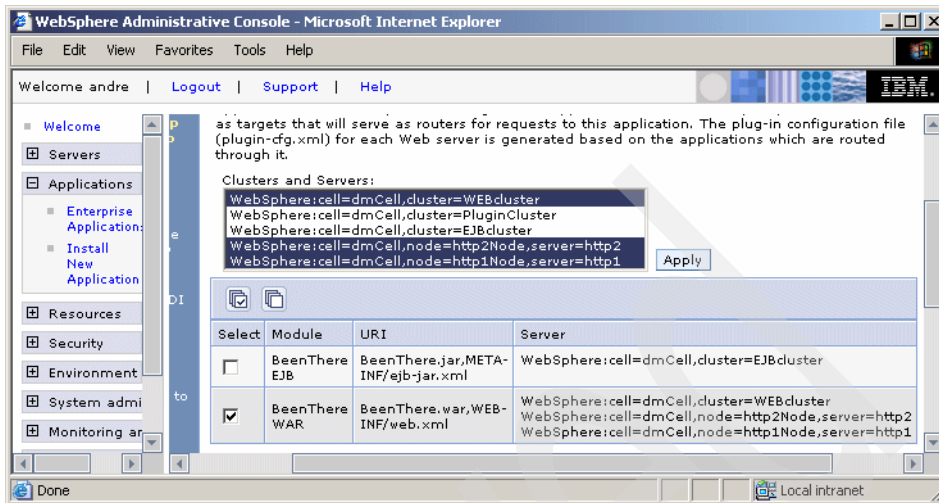


Figure 8-33 Install New Application - Map modules to servers

- d. Accept the defaults on the Provide options to perform the EJB Deploy panel and click **Next**.
- e. Accept the EJB JNDI name supplied in Step 4 (Provide JNDI Names for Beans) and click **Next** to reach Step 5 (Map EJB references to beans).
- f. In this step you must configure an existing EJB reference in the Web module to the BeenThereBean EJB.

Instead of using just the EJB's short JNDI name (as was the case in Step 4 of the installation process) you must enter the fully qualified JNDI name of **cell/clusters/EJBcluster/BeenThere**. This is shown in Figure 8-34.

Map EJB references to beans					
Each Enterprise JavaBeans (EJB) reference that is defined in your application must map to an enterprise bean.					
Module	EJB	URI	Reference binding	Class	
BeenThere.WAR		BeenThere.war,WEB-INF/web.xml	BeenThereBean	com.ibm.websphere.samples.beenthere.BeenThere	cell/clusters/EJBcluster/Been

Figure 8-34 Install New Application - Map EJB references to beans

- g. Click **Next** to reach Step 6 (Map virtual hosts for Web modules). Accept the default virtual host (default_host) and click **Next**.
- h. Step 7 (Map security roles to users/groups) and Step 8 (Ensure all unprotected...) are unnecessary for our configuration, since security is not enabled. Therefore, accept the defaults for both and click **Next**.

- i. The last step shows the installation summary. Review it and click **Finish** to perform the installation. An example of the installation log can be seen in Figure 8-35.

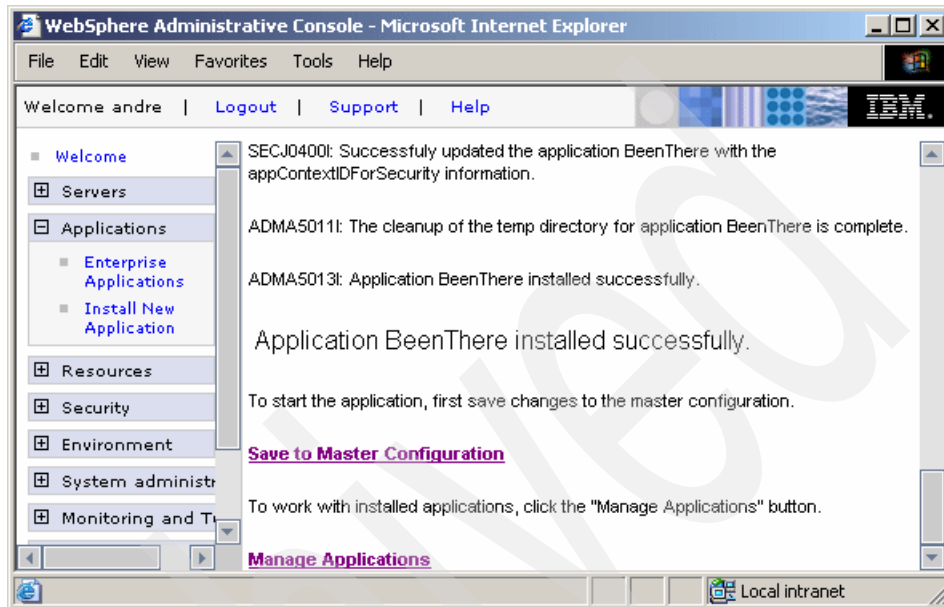


Figure 8-35 Install New Application process - log of actual installation

4. Click **Save to Master Configuration** to save your changes (and do not forget to check the **Synchronize changes with Nodes** option).

8.7.3 Regenerate Web server plug-in

After installing BeenThere, you may need to regenerate the Web servers plug-in configuration files. The plug-in configuration files can also be generated and propagated automatically, if the Web servers' configuration is set up to do so (see 6.5.2, "Generation of the plug-in configuration file" on page 258 for information about this setting).

To manually update and propagate the plug-in file, log on to the WebSphere Administrative Console and select **Servers -> Web servers**, check both **http1** and **http2** Web servers and click the **Generate Plug-in** button.

Alternatively you can regenerate a Web server plug-in from the command line on the Deployment Manager machine using the **GenPluginCfg** script:

```
GenPluginCfg.bat -cell.name dmCell -node.name http1Node -webserver.name http1
```

After the operation completes, the plugin-cfg.xml contains the latest configuration. This file is stored on the Deployment Manager node and then propagated (or copied manually) to the Web server nodes. The plug-in configuration file location on the Deployment Manager node is:

```
<WAS_HOME>/profiles/<dmProfileName>/config/cells/<cellName>/nodes/  
<nodeName>/servers/<serverName>/plugin-cfg.xml
```

Now check both Web servers again and click the **Propagate Plug-in** button to update the plug-in configuration file on both Web servers.

Important: The propagation only works for Web servers on managed nodes or for the IBM HTTP Server 6.0, as explained in 6.5.3, “Propagation of the plug-in file” on page 262.

If propagation of the plug-in file is not possible, you need to copy it manually to the Web servers. The destination path on the Web servers is:

```
<PLUGIN_HOME>/config/<serverName>/plugin-cfg.xml
```

8.7.4 Configuring WEBcluster members for BeenThere

The clusters have already been started but you now need to start the BeenThere application. Go to **Applications -> Enterprise Applications**, select BeenThere and click **Start**. Then open the BeenThere start page using

```
http://<Web_Server_Name>/wlm/BeenThere
```

When you first open BeenThere you will notice the error message shown in Figure 8-36 on page 433:

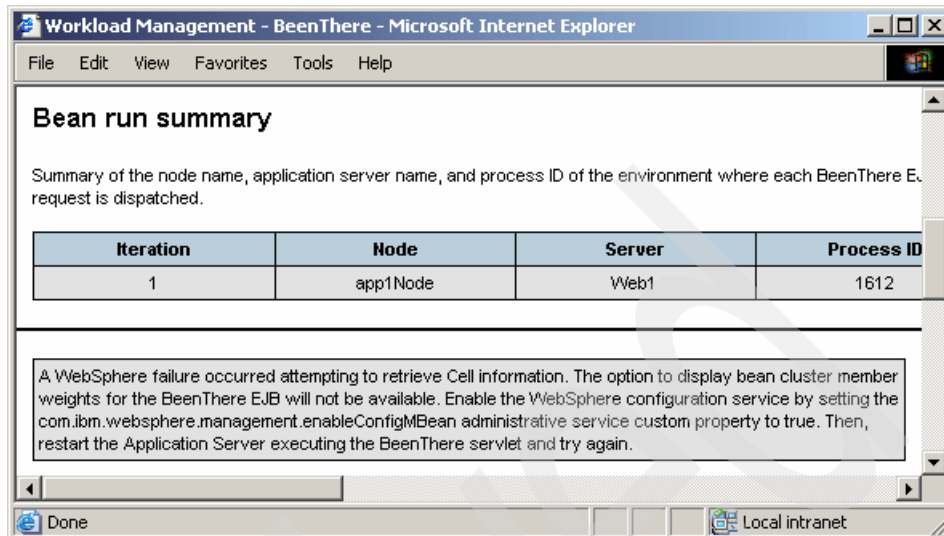


Figure 8-36 BeenThere MBean error message

The BeenThere application requires a rather unusual (but in this case very useful) server configuration in order to display some WebSphere internal data.

Open the WebSphere Administrative Console and select **Servers -> Application servers -> Web1 -> Administration -> Administration Services -> Custom Properties**. Click **New** and add the following custom property:

com.ibm.websphere.management.enableConfigMBean

Enter **true** as the Value. Click **OK**. Your console should now look like Figure 8-37 on page 434.

Repeat this procedure for all other members of the WEBcluster (Web2a and Web2b).

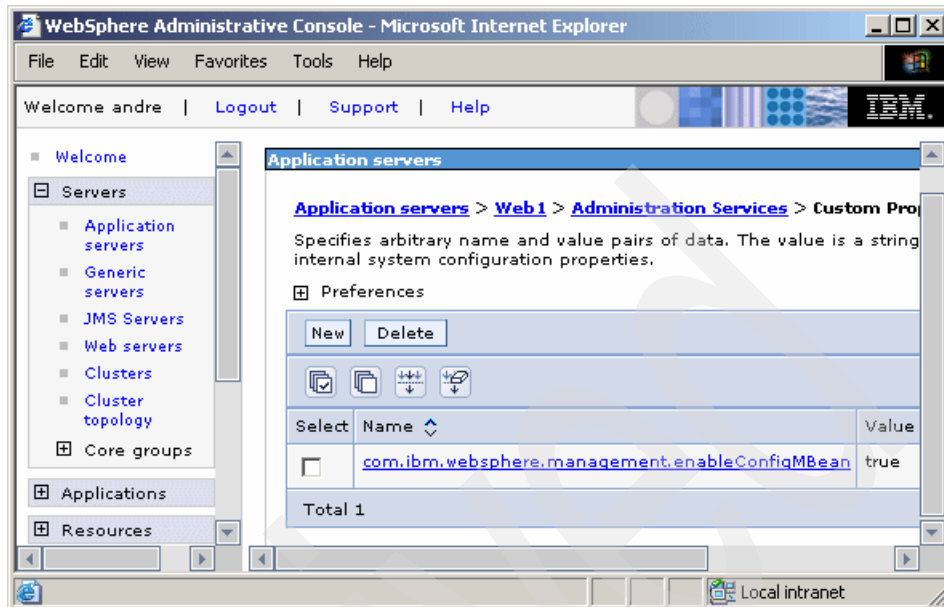


Figure 8-37 Adjusting servers for the BeenThere special requirement

While still logged on to the WebSphere Administrative Console, click **Servers -> Clusters** and **Stop** the cluster called WEBcluster. Once all its members have stopped, select the cluster again and click **Start**. Start the EJBcluster if it is not already started.

Note: There should be no need to restart any other servers other than the WEBcluster members.

8.7.5 Verifying BeenThere

The last step is to verify that BeenThere is working correctly. Follow these steps to verify the configuration:

1. Open the BeenThere start page. Point your browser to the following URL:

`http://<your_caching_proxy>/wlm/BeenThere`

In our environment:

`http://cproxy.itso.ibm.com/wlm/BeenThere`

Notes:

- ▶ cproxy.itso.ibm.com resolves to the IP address of our Caching Proxy (10.20.10.101). The caching proxy then points to the Web server cluster (10.20.10.100) associated with the Load Balancer machine (10.20.10.101).
- ▶ You can also bypass the Caching Proxy and directly use one of the Web servers (http1 or http2) in the URL.
- ▶ If you encounter problems accessing BeenThere through your Web server or Caching Proxy, try to access BeenThere directly via your application server's WebContainer Inbound Chain using:

`http://<your_app_server>:9080/wlm/BeenThere`

In our environment, the URL is:

`http://app1.itso.ibm.com:9080/wlm/BeenThere`

The port number depends on your configuration.

2. If you keep reloading the page you should see that all WEBcluster members are round-robin chosen by the plug-in workload management (see Figure 8-38). As this application does not use sessions, this is the expected behavior.

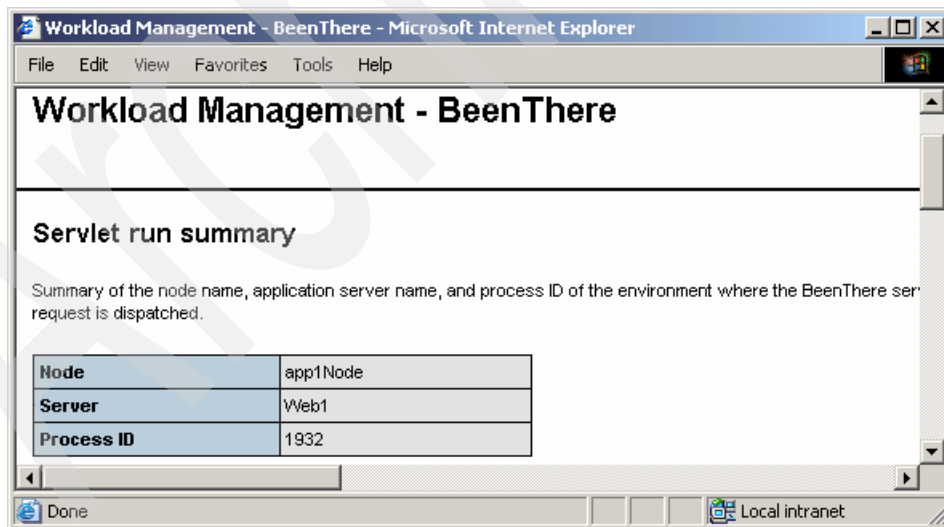


Figure 8-38 BeenThere plug-in workload management

3. Enter **6** for the number of Bean invocations and click **Run**.

The result should look like Figure 8-39. As you can see, the six EJB requests have been workload-managed across the EJB cluster members on both application server nodes app1Node and app2Node. This demonstrates EJB weighted round robin workload management.

Each time you click the **Run** button there is only one request to the Web cluster. The application logic itself invokes the EJB sample method as many times as you asked.

Iteration	Node	Server	Process ID
1	app2Node	EJB2b	19368
2	app2Node	EJB2a	19738
3	app1Node	EJB1	1840
4	app2Node	EJB2b	19368
5	app2Node	EJB2a	19738
6	app1Node	EJB1	1840

Figure 8-39 *BeenThere demonstrating EJB workload management*

You can now play around with the BeenThere application and with the cluster configuration. Using the BeenThere application you can see some run-time statistics and inspect the cluster member weights. Additional things to test are for example:

- ▶ Stopping cluster members to see how the remaining members serve the requests.
- ▶ Change application server weights to see how many requests are served by each member.

8.8 Installing and configuring Trade 6

We now describe how to install and configure the IBM Trade Performance Benchmark Sample for WebSphere Application Server (called Trade 6 throughout this book) into the ITSO sample topology (shown in Figure 8-1 on

page 390). Compared to the lightweight BeenThere application, Trade 6 is much more complex.

Trade 6 is the fourth generation of the WebSphere end-to-end benchmark and performance sample application. The Trade benchmark is designed and developed to cover the significantly expanding programming model and performance technologies associated with WebSphere Application Server. This application provides a real-world workload, enabling performance research and verification test of the Java™ 2 Platform, Enterprise Edition (J2EETM) 1.4 implementation in WebSphere Application Server, including key performance components and features.

Overall, the Trade application is primarily used for performance research on a wide range of software components and platforms. This latest revision of Trade builds off of Trade 3, by moving from the J2EE 1.3 programming model to the J2EE 1.4 model that is supported by WebSphere Application Server V6.0. Trade 6 adds DistributedMapbased data caching in addition to the command bean caching that is used in Trade 3. Otherwise, the implementation and workflow of the Trade application remains unchanged.

Trade's new design enables performance research on J2EE 1.4 including the new Enterprise JavaBeans™ (EJB™) 2.1 component architecture, message-driven beans, transactions (1-phase, 2-phase commit) and Web services (SOAP, WSDL, JAX-RPC, enterprise Web services). Trade 6 also drives key WebSphere Application Server performance components such as dynamic caching, WebSphere Edge Server, and EJB caching.

For details on how Trade 6 exploits all of these functions, please read the document `tradeTech.pdf` which can be found inside the installation package.

Trade 6 demonstrates several new features in WebSphere V6. All Trade-versions are WebSphere-version dependent, so Trade 6 will only work with WebSphere Application Server V6.

Note: The most common configurations for Trade 6 are single server and horizontal clustering scenarios.

For our sample topology, we are using a split-JVM topology which at the same time uses horizontal and vertical clustering. As mentioned earlier, the split-JVM topology is not the best from a performance point of view.

Trade 6.0.1 installation summary

The following actions have to be performed in order to set up and install Trade 6:

- Download the Trade 6.0.1 installation package

- ▶ Set up and configure tradedb database
- ▶ Configure the WebSphere cell
- ▶ Install the application
- ▶ Regenerate the Web server plug-in (plugin-cfg.xml)
- ▶ Restart servers

8.8.1 Download the Trade 6.0.1 installation package

You can download the Trade 6.0.1 installation package from the following Web site:

<http://www.ibm.com/software/webservers/appserv/was/performance.html>

Note: At the time of writing this redbook, Trade 6 was not yet available for download. It is expected soon. Please monitor this page for availability.

After downloading the file `tradeInstall.zip`, unzip the package somewhere on the Deployment Manager machine. A directory called `tradeInstall` is created. This directory contains all Trade 6 files.

8.8.2 Set up and configure tradedb database

Trade 6 uses a database to store its data. We assume the use of DB2/UDB V8.2 here (you could also use an Oracle database).

You have to create a DB2 database called “tradedb” on the database server and DB2 client access must be set up on each application server node.

DB2 Server

We are using the default DB2 instance on the server (db2inst1).

1. In a Windows environment, log on to the database server as the DB2 administrator and start a DB2 shell using the `db2cmd` command.

AIX: On AIX, first switch to user db2inst1:

```
su - db2inst1
```

2. Copy the file `Table.ddl` from the Trade 6 install directory to the DB2 server. Then execute the following DB2 commands to create the tradedb database:

```
db2 create db tradedb
db2 connect to tradedb
db2 -tvf Table.ddl
db2 disconnect all
db2 update db config for tradedb using logfilsiz 1000
```



```
db2 update db cfg for tradedb using maxappls 100
db2stop force
db2start
```

DB2 clients

Because Trade 6 uses the DB2 universal JDBC Type 4 driver, it is not necessary to configure the DB2 clients on the application server systems any more (as was needed for previous versions of Trade). If however, for some reason, you have problems accessing the database, following these steps should solve your connection problems.

Tip: If for some reason your commands do not work, please try them with the double quotation mark, for example:

```
db2 "connect to tradedb user <db_user> using <password>"
```

1. Log on to each of the application server machines (app1, app2, and the Deployment Manager system also) as the DB2 instance owner and start a db2 shell.
2. Using this shell, you can now catalog the remote database server db (thus creating a local node representing it) using the command:

```
db2 catalog tcpip node <your_local_node> remote <db2_server_hostname>
server <port_from_the_services_file_on_your_OS_(50000)>
```

So in our case, the correct command is:

```
db2 catalog tcpip node db remote db server 50000
```

Note: Please note that the host name (db) must be resolvable on the DB2 client machine.

3. Next you have to catalog the remote database tradedb:

```
db2 catalog database tradedb as tradedb at node <your_local_node>
```

So in our case, the correct command is:

```
db2 catalog database tradedb as tradedb at node db
```

4. Verify the database connectivity from the DB2 client machines (app1, app2 and dm):

```
db2 connect to tradedb user <db_user> using <password>
```

In our case, the following command results in displaying the information shown in Example 8-1 on page 440:

```
db2 connect to tradedb user db2inst1 using <password>
```

Database Connection Information

Database server	= DB2/NT 8.2.0
SQL authorization ID	= DB2INST1
Local database alias	= TRADEDDB

8.8.3 Configure the WebSphere cell

In order to prepare your J2EE environment to run Trade 6, you have to create several resources (like JDBC and JMS resources). For example, on each application server node there must exist a JDBC data source that points to the database tradedb you just created.

The configuration steps can be performed using the WebSphere Administrative Console, but they would take a lot of time. Fortunately, however, Trade 6 comes with a .jacl script that can be used to configure and install Trade 6 in a clustered environment. We are using these script files.

Tip: In case you are not familiar with Tcl scripting and .jacl files, you can learn more at <http://sourceforge.net/projects/tcljava>.

1. Open up an OS command prompt on the Deployment Manager machine and go to the Trade 6 code directory.
2. Run the following command:

```
<WAS_HOME>\wsadmin -f trade.jacl configure
```

AIX: On AIX, perform the following steps to run the script:

1. Log on as root user
2. Now run the following command:

```
<WAS_HOME>/wsadmin.sh -f trade.jacl configure
```

In Example 8-2 on page 441 you can see an installation walkthrough for the Windows environment (using the configure option). To make it easier to read, input that the user has to provide, has been bolded.

Important: Due to the design of the Trade application, it is important to configure the resources on the EJBcluster and not on the WEBcluster. This is important because the messaging resources, such as the messaging engines, MDBs, and JMS queues and topics, are predominantly accessed from the EJB container. The Web components can be mapped to the WEBcluster and the Web servers later during application installation, so this is not an issue.

Example 8-2 Trade 6 installation (configure option)

Trade Install/Configuration Script

Operation: configure
Silent: false

Global security is (or will be) enabled (true|false) [false]:

Is this a cluster installation (yes|no) [no]:
yes

Collecting Cluster and Cluster Member Info

Note: Before proceeding, all nodes intended for use in this cluster must be federated with the deployment manager using the addNode command! To ensure that this process goes smoothly, it is also important to verify that each node can ping the other cluster nodes based on the host names configured within the WebSphere profile creation tool.

Have all nodes been federated and network connectivity verified? (yes|no) [yes]:

Please enter the cluster name [TradeCluster]:
EJBcluster

Available Nodes:
app1Node
app2Node
dmNode
http1Node

http2Node

Select the desired node [app1Node]:

Please enter the cluster member name [TradeServer1]:

Ejb1

Current Cluster Nodes and Members:

app1Node - Ejb1

Add more cluster members (yes|no) [yes]:

Available Nodes:

app1Node

app2Node

dmNode

http1Node

http2Node

Select the desired node [app1Node]:

app2Node

Please enter the cluster member name [TradeServer2]:

Ejb2a

Current Cluster Nodes and Members:

app1Node - Ejb1

app2Node - Ejb2a

Add more cluster members (yes|no) [yes]:

Available Nodes:

app1Node

app2Node

dmNode

http1Node

http2Node

Select the desired node [app1Node]:

app2Node

Please enter the cluster member name [TradeServer3]:

Ejb2b

Current Cluster Nodes and Members:

app1Node - Ejb1

```
app2Node - Ejb2a
app2Node - Ejb2b
```

Add more cluster members (yes|no) [yes]:

no

Cluster information obtained...

Collecting Database/Datasource Information

Select the backend database type (db2|oracle) [db2]:

NOTE: wsadmin requires ";" for delimiting the database driver path regardless of platform!

Please enter the database driver path
[c:/sqllib/java/db2jcc.jar;c:/sqllib/java/
db2jcc_license_cu.jar;c:/sqllib/java/db2jcc_license_cisuz.jar]:
\${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar;\${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db
2jcc_license_cu.jar

Please enter the database name [tradedb]:

Please enter the DB2 database hostname [localhost]:

db

Please enter the DB2 database port number [50000]:

Please enter the database username [db2admin]:

db2admin

Please enter the database password [password]:

Configuring Cluster and Cluster Members
Scope: dmCell(cells/dmCell|cell.xml#Cell_1)

Creating Cluster EJBcluster...

EJBcluster already exists!

Creating Cluster Member Ejb1...

Ejb1 already exists!

Enabling SIB Service on Ejb1...
SIB Service enabled successfully!

Creating Cluster Member Ejb2a...
Ejb2a already exists!

Enabling SIB Service on Ejb2a...
SIB Service enabled successfully!

Creating Cluster Member Ejb2b...
Ejb2b already exists!

Enabling SIB Service on Ejb2b...
SIB Service enabled successfully!

Cluster Configuration Completed!!!

Configuring JDBC/Datasource Resources
Scope: dmCell(cells/dmCell|cell.xml#Cell_1)

Creating JAAS AuthData TradeDataSourceAuthData...
Alias Name: TradeDataSourceAuthData
User: db2admin
Password: *****
TradeDataSourceAuthData created successfully!

Creating JDBC Provider DB2 Universal JDBC Driver Provider (XA)...
Provider Name: DB2 Universal JDBC Driver Provider (XA)
Implementation Class: com.ibm.db2.jcc.DB2XADataSource
XA enabled: true
DB2 Universal JDBC Driver Provider (XA) created successfully!

Creating DataSource TradeDataSource...
Datasource Name: TradeDataSource
JNDI Name: jdbc/TradeDataSource
Statement Cache Size: 60
Database Name: tradedb
JDBC Driver Type: 4
Hostname: db
Port Number: 50000

Creating Datasource properties...
Creating Connection Pool using defaults...
Creating Connection Factory...
TradeDataSource created successfully!

```
Creating JDBC Provider DB2 Universal JDBC Driver Provider...
  Provider Name:      DB2 Universal JDBC Driver Provider
  Implementation Class: com.ibm.db2.jcc.DB2ConnectionPoolDataSource
  XA enabled:        false
DB2 Universal JDBC Driver Provider created successfully!
```

```
Creating DataSource MEDataSource...
  Datasource Name:    MEDataSource
  JNDI Name:          jdbc/MEDataSource
  Statement Cache Size: 60
  Database Name:      tradedb
  JDBC Driver Type:   4
  Hostname:           db
  Port Number:        50000
```

```
Creating Datasource properties...
Creating Connection Pool using defaults...
Creating Connection Factory...
MEDataSource created successfully!
```

```
-----
JDBC Resource Configuration Completed!!!
-----
```

```
-----
Configuring JMS Resources
Scope: dmCell(cells/dmCell|cell.xml#Cell_1)
-----
```

```
Creating JAAS AuthData TradeOSUserIDAuthData...
  Alias Name: TradeOSUserIDAuthData
  User:      LocalOSUserID
  Password:  password
TradeOSUserIDAuthData created successfully!
```

```
Creating SIBus EJBcluster...
EJBcluster created successfully!
```

```
Adding SIBus member EJBcluster...
  Default DataSource: false
  Datasource JNDI Name: jdbc/MEDataSource
SIBus member added successfully!
```

```
Creating SIB Messaging Engine...
  Bus Name:      EJBcluster
  Default DataSource: false
  Datasource JNDI Name: jdbc/MEDataSource
  Cluster Name:  EJBcluster
```

created successfully!

Creating SIB Messaging Engine...

Bus Name: EJBcluster
Default DataSource: false
Datasource JNDI Name: jdbc/MEDataSource
Cluster Name: EJBcluster
created successfully!

Creating OneOfNPolicy Policy for ME0...

Alive Period(s): 30
Server Name: Ejb1
ME Name: EJBcluster.000-EJBcluster
Policy for ME0 created successfully!

Modifying ME DataStore parameters...

ME Name: EJBcluster.000-EJBcluster
AuthAlias: TradeDataSourceAuthData
Schema Name: IBMME0
EJBcluster.000-EJBcluster data store modified successfully!

Creating OneOfNPolicy Policy for ME1...

Alive Period(s): 30
Server Name: Ejb2a
ME Name: EJBcluster.001-EJBcluster
Policy for ME1 created successfully!

Modifying ME DataStore parameters...

ME Name: EJBcluster.001-EJBcluster
AuthAlias: TradeDataSourceAuthData
Schema Name: IBMME1
EJBcluster.001-EJBcluster data store modified successfully!

Creating OneOfNPolicy Policy for ME2...

Alive Period(s): 30
Server Name: Ejb2b
ME Name: EJBcluster.002-EJBcluster
Policy for ME2 created successfully!

Modifying ME DataStore parameters...

ME Name: EJBcluster.002-EJBcluster
AuthAlias: TradeDataSourceAuthData
Schema Name: IBMME2
EJBcluster.002-EJBcluster data store modified successfully!

Creating SIB Destination TradeBrokerJSD...

Destination Name: TradeBrokerJSD
Destination Type: Queue
Reliability: EXPRESS_NONPERSISTENT


```
Cluster Name:      EJBcluster
TradeBrokerJSD created successfully!

Creating SIB Destination Trade.Topic.Space...
Destination Name:  Trade.Topic.Space
Destination Type:  TopicSpace
Reliability:       EXPRESS_NONPERSISTENT
Trade.Topic.Space created successfully!

Creating JMS Queue Connection Factory TradeBrokerQCF...
Connection Factory Name: TradeBrokerQCF
Connection Factory Type: Queue
JNDI Name:          jms/TradeBrokerQCF
TradeBrokerQCF created successfully!

Creating JMS Topic Connection Factory TradeStreamerTCF...
Connection Factory Name: TradeStreamerTCF
Connection Factory Type: Topic
JNDI Name:          jms/TradeStreamerTCF
TradeStreamerTCF created successfully!

Creating JMS Queue TradeBrokerQueue...
Queue Name:        TradeBrokerQueue
JNDI Name:         jms/TradeBrokerQueue
SIB Destination:   TradeBrokerJSD
Delivery Mode:     NonPersistent
TradeBrokerQueue created successfully!

Creating JMS Topic TradeStreamerTopic...
Topic Name:        TradeStreamerTopic
JNDI Name:         jms/TradeStreamerTopic
Topic Space:       Trade.Topic.Space
Delivery Mode:     NonPersistent
TradeStreamerTopic created successfully!

Creating MDB Activation Spec TradeBrokerMDB...
MDB Activation Spec Name: TradeBrokerMDB
JNDI Name:           eis/TradeBrokerMDB
JMS Destination JNDI Name: jms/TradeBrokerQueue
Destination Type:     javax.jms.Queue
TradeBrokerMDB created successfully!

Creating MDB Activation Spec TradeStreamerMDB...
MDB Activation Spec Name: TradeStreamerMDB
JNDI Name:           eis/TradeStreamerMDB
JMS Destination JNDI Name: jms/TradeStreamerTopic
Destination Type:     javax.jms.Topic
TradeStreamerMDB created successfully!
```

JMS Resource Configuration Completed!!!

Saving...

Saving config...

Important: Because we have used a variable for the JDBC driver path, we now need to set the value for this variable according to our DB2 client installation. Go to **Environment -> WebSphere Variables**. Click **DB2UNIVERSAL_JDBC_DRIVER_PATH** and enter the path to your SQLLIB/java directory, for example C:/Program Files/IBM/SQLLIB/java. You need to set this value for each node individually or you can use a cell wide variable if the path is identical on all nodes.

To use a cell wide variable, you must delete any existing DB2UNIVERSAL_JDBC_DRIVER_PATH variable (which, by default, exists but contains no value) on the nodes. If the variable exists at the node level, the cell wide variable will not be used and you will not be able to connect to the database if there is no or a wrong value provided in the node level variable.

Well, that was long. Therefore, we now explain the individual steps taken by this script:

► Collecting Cluster and cluster member Info

As the first task, the script collects all data regarding the desired topology (cluster, nodes and cluster members). As shown in the walkthrough, you must enter our sample topology configuration for the cluster EJBcluster.

► Collecting database/datasource Information

The script now collects information regarding the database connection. You must supply the same data as when configuring (and testing) the DB2 client on each node. One thing that deserves special attention is the use of a WebSphere variable (DB2UNIVERSAL_JDBC_DRIVER_PATH) on the path to DB2 java libraries (JARs).

Attention: As you will see, the script creates cell wide datasource definitions. That is why the use of WebSphere variables is the only way that cell-level JDBC data providers can work in a heterogeneous environment.

You can define a default cell-level value for the variable used here (like "C:\SQLLIB\java"), and override it with a different value (like "/home/db2inst1/sqllib/java") at node level when necessary.

- ▶ Configuring the cluster and cluster members

Here the script tries to create the cluster and its members (if they are not already created). The script also enables the Service Integration Bus service at server startup.

- ▶ Configuring JDBC/datasource resources

Now the script creates the J2C authentication entry (TradeDataSourceAuthData) which is needed when security is enabled, the JDBC providers and their data sources. There is one non-XA data source ("MEDataSource", to be used by the messaging engines) and one XA data source ("TradeDataSource", to be used by the application).

- ▶ Configuring JMS resources

Here the script creates the required J2C authentication entry (TradeOSUserIDAuthData), creates a bus (also named EJBcluster) and sets the EJBcluster cluster as its member.

Then the script creates two additional messaging engines (one was created automatically). Each ME gets its own schema name (in other words, each ME will have its own set of tables in the database) and a policy that locks it in a single server.

Note: In WebSphere Application Server V6, a ME is an in-process service running within an Application Server. HAManager's default behavior makes every ME able to "float" between a list of servers (you can even state the preferred ones). So you could have a single ME that automatically starts up on a different cluster member whenever its hosting server is no longer available. The best performance, though, is obtained by having an ME in every server. The mentioned policies restrain the ME's ability to start up on another server.

Finally the script creates several messaging-related resources, such as bus destinations, JMS queues/topics (and connection factories), and JMS activation specification for the MDBs (message-driven beans).

- ▶ Saving configuration

The final step is that all changes are saved into the cell's master configuration.

8.8.4 Install Trade 6 from the WebSphere Administrative Console

You are now ready to install the Trade 6 Enterprise Application Archive. We could install Trade 6 by using the script again (this time with the install option). Instead, we shall do it using the WebSphere Administrative Console.

Note: The reason for installing Trade6.ear using the Administrative Console rather than the .jacl script is our split-JVM environment. Using the Administrative Console allows you to deploy the Web components to the WEBcluster and the EJBs to the EJBcluster during the installation. The alternative is to install the application using the script and then change the mappings to the clusters and the EJB references afterwards.

If you do not use such an environment, you can easily use the .jacl script for the install process. In fact, you can then actually configure the environment and install the application at the same time - in one call of the script.

See 8.8.6, “Install Trade 6 using the installation script” on page 453 for information about how to use the script for the install process.

1. Log on to the WebSphere Administrative Console and click **Applications** -> **Install New Application**. The Preparing for the application installation window is displayed (compare to the BeenThere installation, Figure 8-31 on page 428).
 - a. First you have to specify the EAR/WAR/JAR module to upload and install. Select **Browse** to specify the location of the Trade.ear file, select it, and click **Next** to continue.
- Note:** The .ear file can reside on any managed node in the cell or on the local workstation running the WebSphere Administrative Console. Depending on where you stored it, click the Browse button next to the Local file system or Server file system path.
- b. On the next window you can define mappings and bindings. We do not need to change anything here, so click **Next** to accept all defaults.
 - c. Click the **Continue** button on the Application Security Warnings window.
 2. The upcoming window shows the first of 12 steps in the Install New Application process.
 - a. On the Step 1 window (Select installation options), make sure the **Deploy enterprise beans** check box is selected. Please note that the Application Name Trade has been filled in automatically. Click **Next**.

- b. On the Step 2 window (Map modules to servers) change the default mapping so that the **EJB module** (TradeEJBs) is mapped to **EJBcluster** and the **Web module** is mapped to **WEBcluster** as well as to both **Web servers** (just like you did during the BeenThere installation, see Figure 8-33 on page 430). Click **Next**.
- c. On the Step 3 window (Provide options to perform the EJB Deploy) you should change the Database type to the version you are using (in our case to **DB2UDB_V82**).
- d. Click **Step 9** (Map EJB references to beans). A window similar to Figure 8-40 on page 452 is shown.
 - i. Change the JNDI name of the ejb/Trade reference binding in the TradeWeb module to its fully qualified JNDI name:
`cell/clusters/<EJB_Cluster_Name>/ejb/TradeEJB`
In our environment, the correct fully qualified JNDI name is:
`cell/clusters/EJBcluster/ejb/TradeEJB`
 - ii. Change the JNDI name of ejb/Quote reference binding in the TradeWeb module to its fully qualified JNDI name:
`cell/clusters/<EJB_Cluster_Name>/ejb/QuoteEJB`
In our environment, the correct fully qualified JNDI name is:
`cell/clusters/EJBcluster/ejb/QuoteEJB`

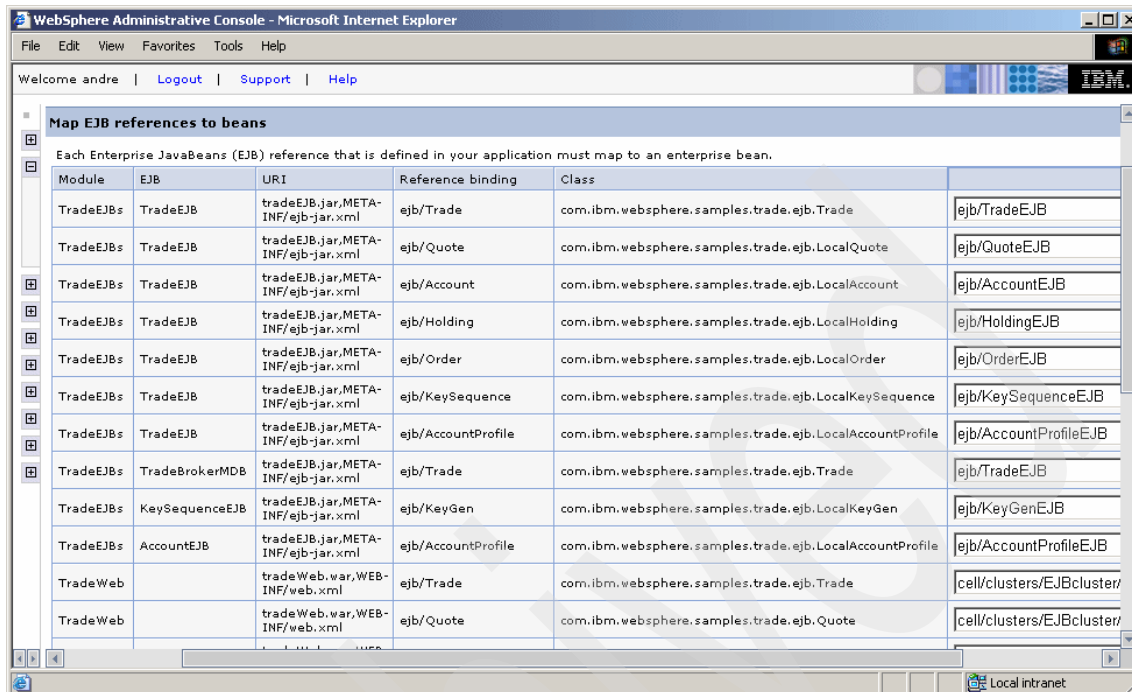


Figure 8-40 Provide fully qualified JNDI names for the TradeWeb module level

- e. Click **Step 12 (Summary)**. Review the options and click **Finish** to perform the installation (see Figure 8-41 on page 453). This can take a few minutes.

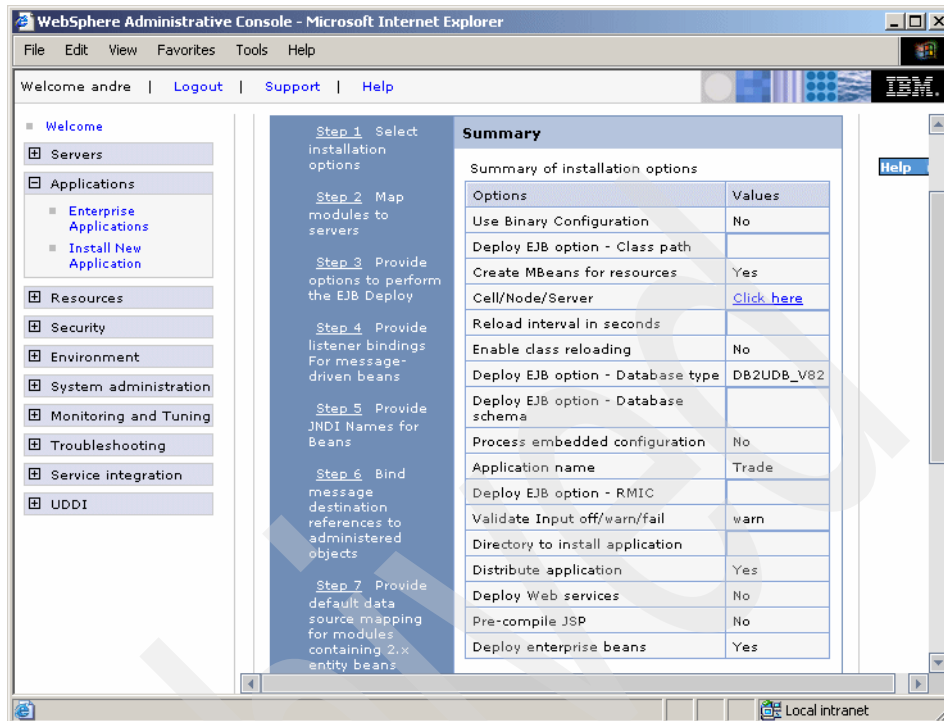


Figure 8-41 Summary for installing the Trade 6 application

3. Finally, after the installation has completed, save your changes to the master configuration and synchronize with the nodes.

8.8.5 Regenerate Web server plug-in and start servers

After installing Trade 6 you may need to regenerate and propagate (or copy) the Web servers' plug-in configuration files. The procedure is the same as shown in "Regenerate Web server plug-in" on page 431.

After the regeneration and propagation of the plug-in is finished, you have to restart the clusters (WEBcluster and EJBcluster).

8.8.6 Install Trade 6 using the installation script

You can alternatively use the installation script that comes with Trade 6 to configure and install it into a cluster. However, this installation script installs Trade6 EAR into a non-split-JVM environment (which is recommended anyway for performance reasons).

If you want to use the install script for Web and EJB containers in different clusters, then you need to manually change the mappings after installation. This is explained in step 6 on page 457.

To perform the installation and configuration using the trade.jacl script follow these steps:

1. Open an OS command prompt on the Deployment Manager node and go to the directory where the tradeInstall.zip files were extracted to (for example "C:\tradeInstall" or "/tmp/tradeInstall").
2. Run the WebSphere Application Server Network Deployment **setupCmdLine** script to setup your shell environment.

```
C:\tradeInstall>C:\WebSphere\AppServer\bin\setupCmdLine.bat
```

AIX: On AIX, do the following:

```
/tmp/tradeInstall # /usr/WebSphere/AppServer/bin/setupCmdLine.sh
```

3. Make sure the Deployment Manager is running. If not, you can start it using the **startManager** script:

```
C:\tradeInstall>C:\WebSphere\AppServer\bin\startManager.bat
```

AIX: On AIX, do the following:

```
/tmp/tradeInstall # /usr/WebSphere/AppServer/bin/startManager.sh
```

4. You have two choices:
 - a. Launching the script without specifying an option or using the **all** option, which configures and installs Trade 6 at the same time.

```
C:\tradeInstall>C:\WebSphere\Appserver\bin\wsadmin.bat -f trade.jacl  
all
```

- b. Alternatively, if you have already configured the environment previously using the script with the **configure** option, then you can now only install the code by specifying the **install** option.

```
C:\tradeInstall>C:\WebSphere\AppServer\bin\wsadmin.bat -f trade.jacl  
install
```

AIX: On AIX, use the following command:

```
/tmp/tradeInstall # /usr/WebSphere/AppServer/bin/wsadmin.sh -f  
trade.jacl install (or all)
```


In either case, a dialog launches where you need to enter several settings. When using the `all` option, then you need to enter all the values as described in 8.8.3, “Configure the WebSphere cell” on page 440.

For the install-only process, the following values are needed:

- a. Is this a cluster installation: `yes`
 - b. Enter the cluster name: `EJBcluster`
 - c. Enter the database type: `db2`
5. Wait until the script ends (it takes some time, especially for the EJB deployment). An example of `trade.jacl` script execution (with the `install` option) can be seen in Example 8-3:

Example 8-3 Trade 6 installation (install option)

Trade Install/Configuration Script

Operation: `install`
Silent: `false`

Installing Trade

Is this a cluster installation (yes|no) [no]:
yes

Have all nodes been federated and network connectivity
verified? (yes|no) [yes]:

Please enter the cluster name [TradeCluster]:
EJBcluster

.....
.....

Select the backend database type (db2|oracle) [db2]:

Installing application {Trade}...
Application Name: `Trade`
Ear file: `trade.ear`
Target Cluster: `EJBcluster`
Deploy EJB: `true`
Deploy WebServices: `true`
Use default bindings: `true`
Use Ear MetaData: `true`

```

    Deployed DB Type:      DB2UDB_V82
Starting application install...
WASX7327I: Contents of was.policy file:
    grant codeBase "file:${application}" {
        permission java.security.AllPermission;
    };
Retrieving document at 'file:...../TradeServices.wsdl'.
ADMA5016I: Installation of Trade started.
ADMA5058I: Application and module versions validated with versions of
deployment targets.
ADMA5018I: The EJBDeploy command is running on enterprise archive (EAR) file
C:\.....\app5808.ear.
Starting workbench.
Creating the project.
Building: /tradeEJB
Deploying jar tradeEJB
Creating Top Down Map
Generating deployment code
Refreshing: /tradeEJB/ejbModule.
Building: /tradeEJB
Invoking RMIC.
Generating DDL
Generating DDL
Writing output file
Shutting down workbench.
EJBDeploy complete.
0 Errors, 0 Warnings, 0 Informational Messages
ADMA5007I: The EJBDeploy command completed on
C:\.....\app_fff1dabc32\dpl\dpl_Trade.ear
WSWS0041I: Web services deploy task completed successfully.
ADMA5005I: The application Trade is configured in the WebSphere Application
Server repository.
ADMA5053I: The library references for the installed optional package are
created.
ADMA5005I: The application Trade is configured in the WebSphere Application
Server repository.
ADMA5001I: The application binaries are saved in
C:\WebSphere\AppServer/profiles/dm\wstemp\Scriptfff1d6f38a\workspace\cells\dmCe
11\applications\Trade.ear\Trade.ear
ADMA5005I: The application Trade is configured in the WebSphere Application
Server repository.
SECJ0400I: Successfully updated the application Trade with the
appContextIDForSecurity information.
ADMA5011I: The cleanup of the temp directory for application Trade is complete.
ADMA5013I: Application Trade installed successfully.
Install completed successfully!

```

Trade Installation Completed!!!

Saving...

Saving config...

6. As we are using a split-JVM environment, we must now correct the application server mappings so that the EJBs are deployed onto the EJBcluster and the Web module is mapped to the Web servers also. Open the WebSphere Administrative Console and select **Applications -> Enterprise Applications -> trade -> Map modules to servers** and do the proper mappings.
7. Correct the EJB references in the Web modules, as explained in step 2d on page 451.
8. Save and synchronize the changes, regenerate the plug-in files and propagate/copy them.
9. Verify that both clusters (WEBcluster and EJBcluster) as well as the Trade 6 application are started.

Tip: The Trade 6 configure option did many changes to the environment, so it is a good idea to check the JVM log files for unusual error messages.

8.8.7 Working with Trade 6

The last step is to verify the Trade 6 installation. First you should test if Trade 6 runs on your application server without using the entire topology, that is connecting directly to the WebContainer Inbound Chain of one of your application servers.

Open the Trade 6 start page using the following URL (the port number depends on your configuration):

`http://<host_name>:9080/trade`

In our environment, the URL is:

`http://app1.itso.ibm.com:9080/trade`

If this works, then in a second step you should test using one of the Web servers, for example:

`http://http1.itso.ibm.com/trade`

If this works, then you can test the whole topology, that is, through the Caching Proxy, Load Balancer, and Web server cluster using:

`http://<your_caching_proxy>/trade`

In our environment, the URL is:

`http://cproxy.itso.ibm.com/trade/`

Note: `cproxy.itso.ibm.com` resolves to the IP address of our caching proxy (10.20.10.101). The caching proxy then points to the Web server cluster (10.20.10.100) associated with the Load Balancer machine (10.20.10.101).

All of these URLs display the Trade 6 index page as shown in Figure 8-42.

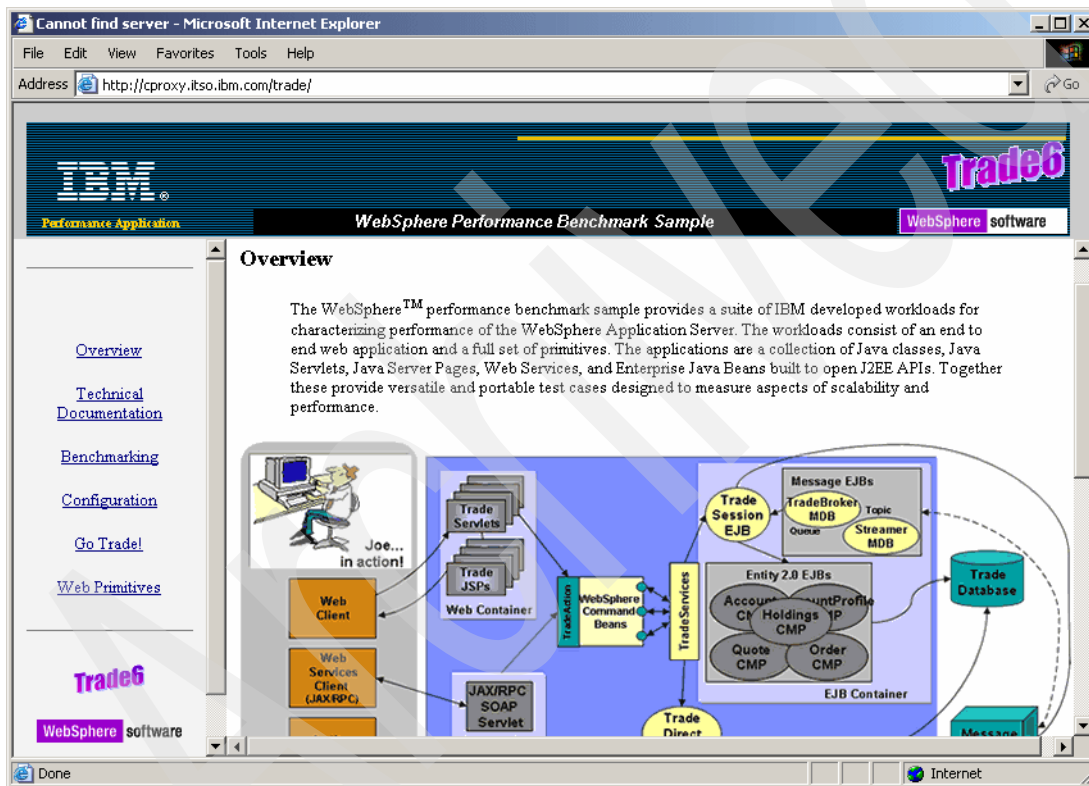


Figure 8-42 Trade 6 index page

Before you can start using the Trade 6 application, you first have to populate the database:

1. Click the **Configuration** link on the left-hand side of the Trade 6 index page.
2. In the Configuration utilities window, click **(Re-)populate Trade Database**. Wait until the database population has been completed.

- Before closing the window, review the default Trade 6 configuration and click the **Update Config** button.
- After populating the database, run the following DB2 commands to update DB2 statistics:

```
db2 connect to tradedb user <db_user> using <password>
db2 reorgchk update statistics
```

Note: It is very important that you run these commands.

- Click the **Go Trade!** link on the left-hand side of the Trade 6 index page. You are forwarded to the Trade 6 login page shown in Figure 8-43.
- Click the **Log in** button and start to use Trade 6.

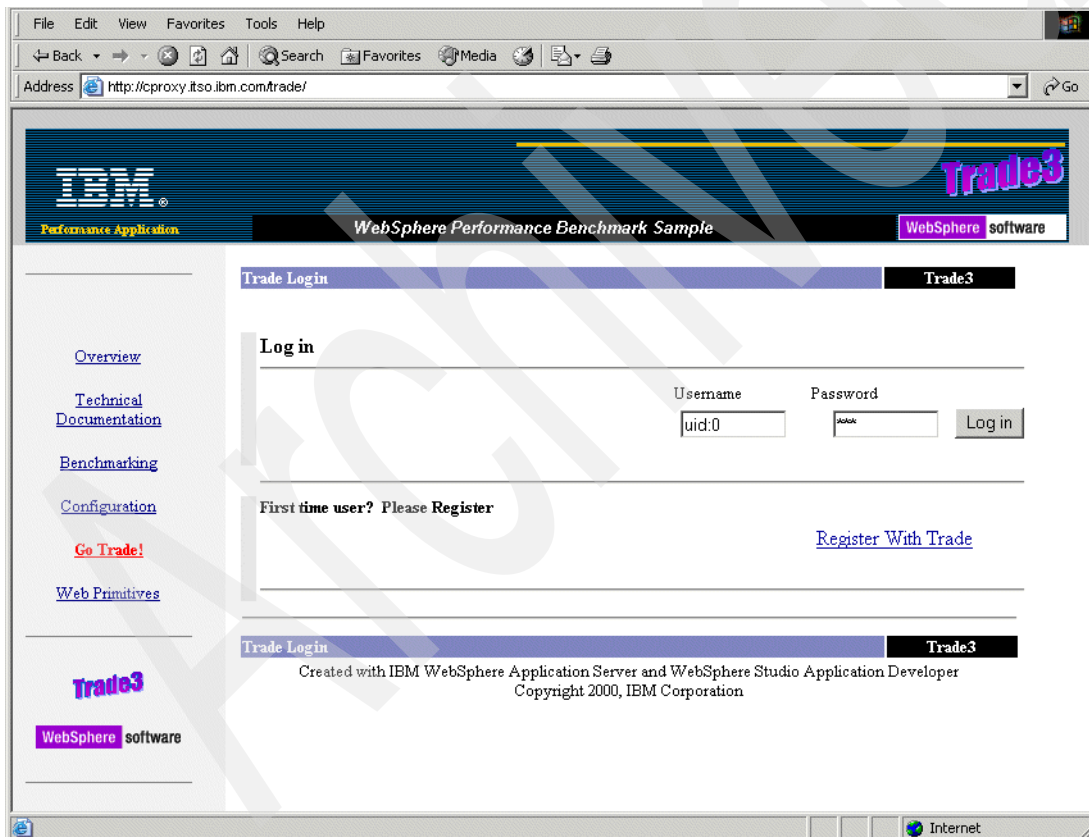


Figure 8-43 Trade 6 Login page

8.8.8 Verify failover with Trade 6

Now that you have Trade 6 up and running, verify WebSphere's failover functionality. The following outlines a scenario you could use to do so. Before you begin, ensure that all servers are started.

1. Log into Trade 6.
2. On one of the WEBcluster servers (Web1, Web2b, or Web2a) a session has been created. Stop two of the Web servers, for example Web1 and Web2b.
3. Continue using Trade 6, this will force Web2a to handle all your application requests. You are now certain that your session lives in the Web2a server.
4. To verify WebSphere failover, you can now restart another Web server of the WEBcluster, for example Web1. After this server has been restarted, you must then stop Web2a.
5. After it has stopped, continue using Trade 6, your session will still exist and you can continue using Trade 6 without any interruptions.

This proves that session persistence is working on the WEBcluster. Another scenario could be to stop one or more EJBcluster application servers, or you could try shutting down an entire node.

8.8.9 Volume testing Trade 6

In order to test the throughput and scalability of a particular application or hardware architecture, you need to simulate a high workload. There are a number of tools available for this purpose. Some are available for free and some are not. Refer to 17.3, "Tools of the trade" on page 945 for instructions on how to do this.

8.8.10 Uninstalling Trade 6

Trade 6 can always be uninstalled like any other enterprise application. However, if you wish to undo all settings made by the trade.jacl script you need to perform these additional steps after removing the application:

1. Write down all schema names used for each of the three messaging engines (you will have to drop all their tables after all following steps). Select **Service integration -> Buses -> EJBcluster -> Messaging engines**. For each messaging engine, click its link, select **Additional Properties -> Data store** and write down the value of the Schema name field (the values should be IBMME0, IBMME1 and IBMME2).
2. Remove the bus named EJBcluster (select **Service integration -> Buses**, check the **WEBcluster** bus and click **Delete**).

3. Remove the data sources:
 - a. First, select **Resources -> JDBC Providers**, choose the **dmCell** scope (by removing the Node name) and click **Apply**. Now select **DB2 Universal JDBC Driver Provider -> Data sources**, select the **MEDataSource** data source and click **Delete**.
 - b. Now select **Resources -> JDBC Providers -> DB2 Universal JDBC Driver Provider (XA) -> Data sources**, check **TradeDataSource** and click **Delete**.
 - c. You can also delete the J2C authentication data entry by selecting **Security -> Global security -> JAAS Configuration -> J2C Authentication data**, checking both the **TradeDataSourceAuthData** and **TradeOSUserIDAuthData** entries and clicking **Delete**. There is no need to remove the JDBC providers.
4. Remove JMS resources. Select **Resources -> JMS Providers -> Default messaging**. This is the Default messaging provider page. Make sure you are at dmCell scope before proceeding.
 - a. Select **JMS queue connection factory** under Connection Factories, check the **TradeBrokerQCF** connection factory and click **Delete**.
 - b. Go back to the Default messaging provider page, select **JMS topic connection factory** from the Connection Factories, check the **TradeStreamerTCF** connection factory and click **Delete**.
 - c. Go back to the Default messaging provider page, under Destinations select **JMS queue**, check the **TradeBrokerQueue** queue and click **Delete**.
 - d. Go back to the Default messaging provider page, select **JMS topic** from the Destinations, check the **TradeStreamerTopic** topic and click **Delete**.
 - e. Go back to the Default messaging provider page, select **JMS activation specification** from the Activation Specifications, check both the **TradeBrokerMDB** and **TradeStreamerMDB** entries and click **Delete**.
5. Remove the core group policies. Select **Servers -> Core groups -> Core group settings -> DefaultCoreGroup -> Policies**, check all three **Policy for MEx** policies and click **Delete**.
6. Finally, save and synchronize the changes. It is a good idea to restart both clusters (WEBcluster and EJBcluster) and the Deployment Manager.
7. No it is not over yet. The last step is to drop all tables that were created for the messaging engines. The schemas created on the database were IBMME0, IBMME1 and IBMME2. Connect to the tradedb database and execute the db2 commands shown in Example 8-4 on page 462.

It is important that you drop all tables listed, and repeat this for each schema.

Example 8-4 Dropping all tables from ME schemas

```
C:\SQLLIB\BIN>db2 connect to tradedb user administrator
Enter current password for administrator:
```

Database Connection Information

```
Database server      = DB2/NT 8.2.0
SQL authorization ID = ADMINIST...
Local database alias = TRADEDDB
```

```
C:\SQLLIB\BIN>db2 list tables for schema ibmme0
```

Table/View	Schema (...)
SIB000	IBMME0
SIB001	IBMME0
SIB002	IBMME0
SIBCLASSMAP	IBMME0
SIBKEYS	IBMME0
SIBLISTING	IBMME0
SIBOWNER	IBMME0
SIBXACTS	IBMME0

8 record(s) selected.

```
C:\SQLLIB\BIN>db2 drop table ibmme0.sib000
```

Attention: It is very important to drop these tables from the database, or else whenever you run the `trade.jacl` configure script again, the MEs created will always fail during instantiation (their UUID will conflict with whatever was left from the previous installation in the database).

More information can be found in the WebSphere Application Server V6 InfoCenter. Search for the sections “Tips for troubleshooting messaging engines” or “Problems when re-creating a Service Integration Bus”.

<http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp>

Lots of information about buses, messaging engines, and data stores can be found in Chapters 10 and 11 of the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.



Part 4

High availability and caching

Architect

WebSphere HAManager

IBM WebSphere Application Server Network Deployment V6 introduces a new feature called High Availability Manager (HAManager) that enhances the availability of WebSphere singleton services like transaction or messaging services and provides an extremely fast native message transport that is leveraged by DRS for both HTTP session replication and stateful session bean failover. It can leverage the latest storage technologies to provide fast recovery time of two-phase transactions. In addition, it adds the possibility of hot standby support to high availability solutions using conventional failover middleware such as IBM HACMP or Tivoli System Automation. We discuss various HA scenarios in this chapter.

Note: There is a major change in this redbook compared to the previous version (*IBM WebSphere V5.1 Performance, Scalability, and High Availability, WebSphere Handbook Series*, SG24-6198-01): we have removed the high availability section of the book and moved it into a new redbook entitled *IBM WebSphere Application Server Network Deployment V6: High availability solutions*, SG24-6688. Therefore, you will find information regarding the new High Availability Manager in this book but no explanation as to how to make WebSphere objects highly available using external clustering software, such as IBM HACMP and Tivoli System Automation. For this kind of information, please refer to the aforementioned redbook.

9.1 Introduction

IBM WebSphere Application Server Network Deployment V6 comes with a new feature: High Availability Manager, commonly called HAManager, to enhance the availability of singleton services in WebSphere as well as provide group services and group messaging capabilities to WebSphere internal components. These singleton services include:

- ▶ Transaction service (Transaction log recovery)
- ▶ Messaging service (Messaging engine restarting)

The HAManager runs as a service within each application server process (Deployment Manager, Node Agents, or application servers) that monitors the health of WebSphere clusters. In the event of a server failure, the HAManager will failover any singleton services that were running on the server that just failed. Examples of such services would include the recovery of any in-flight transactions and/or restarting any messaging engines that were running there. As depicted in Figure 9-1 on page 466, each application server process runs a HAManager component and shares information through the underlying communication infrastructure Distribution and Consistency Services (DCS) such that no single point of failure will exist in the topology. Every member in a WebSphere cluster knows where singleton services are running.

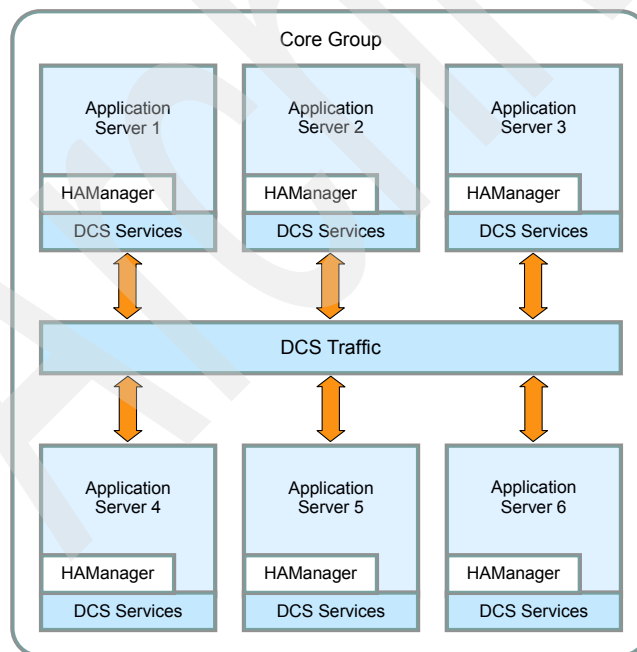


Figure 9-1 Architecture of a single core group configuration

WebSphere V6 uses a peer-to-peer hot failover model that dramatically improves recovery time. By leveraging Network Attached Storage technology, WebSphere clusters can be made highly available with a simpler setup. No external high availability software is required.

Note: Deployment Managers and Node Agents cannot be made highly available with HAManager. Refer to the redbook *WebSphere Application Server Network Deployment V6: High availability solutions*, SG24-6688 for information about how to achieve this.

9.2 Core group

A core group is a high availability domain within a cell. It serves as a physical grouping of JVMs in a cell that are candidates to host singleton services. It can contain stand-alone servers, cluster members, Node Agents or the Deployment Manager, each of these run in a JVM. See Figure 9-2 on page 467.

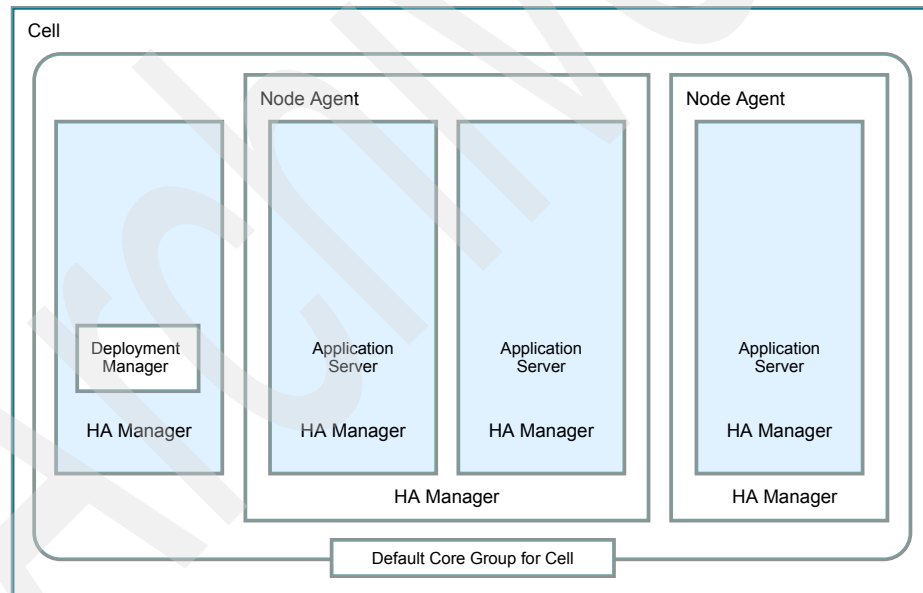


Figure 9-2 Conceptual diagram of a core group

Each JVM process can only be a member of one core group. Naturally, cluster members must belong to the same core group. At runtime, the core group and policy configurations are matched together to form high availability groups. For

more information about policies and high availability groups, see 9.2.4, “Core group policy” on page 474 and 9.3, “High availability group” on page 479.

A set of JVMs can work together as a group to host a highly available service. All JVMs with the potential to host the service join the group when they start. If the singleton is scoped to a WebSphere cluster, such as a Transaction Manager or a messaging engine, then all members of the cluster are part of such a group of JVMs that can host the service. WLM information for clusters is available to all members of the core group in a peer-to-peer fashion.

A core group cannot extend beyond a cell, or overlap with other core groups. Core groups in the same cell or from different cells, however, can be federated to share WLM information using the core group bridge service. In a large-scale implementation with clusters spanning multiple geographies, you can use a variety of transport types for different core groups and link them together with the core group bridge service to form flexible topologies. The most important thing is that all JVMs in a core group must be able to open a connection and send heartbeat messages to each other.

Tip: If your node is installed on a server with IBM HACMP or Tivoli System Automation, you may wish to use the persistent node IP label as the host address for your node because this IP alias is available at boot time whether a HACMP cluster is online or not, and it is also not prone to physical network adapter failures.

9.2.1 Core group coordinator

After the membership of the core group stabilizes at runtime, certain members are elected to act as coordinators for the core group. A core group coordinator is responsible for managing the high availability groups within a core group. The following aspects are managed by the core group coordinator:

- ▶ Maintaining all group information including the group name, group members and the policy of the group.
- ▶ Keeping track of the states of group members as they start, stop, or fail and communicating that to every member.
- ▶ Assigning singleton services to group members and handling failover of services based on core group policies.

Preferred coordinator servers

By default, the HAManager elects the lexically lowest named group member to be the coordinator. Since being a coordinator takes up additional resources in the JVM, you may wish to override the default election mechanism by providing

your own list of preferred coordinator servers in the WebSphere Administrative Console. You can do this by selecting **Servers -> Core groups -> Core group settings -> <core_group_name> -> Preferred coordinator servers**.

Specifying just one server in the list does not make it a single point of failure. The HAManager simply gives the server a higher priority over others in the core group instead of giving it an exclusive right to be a coordinator. Consider the following when deciding which JVMs should become preferred coordinators:

- ▶ Find a JVM with enough memory

Being a coordinator, a JVM needs extra memory for the group and status information mentioned earlier. For core groups of moderate size, the memory footprint is small. You can enable verbose garbage collection to monitor the heap usage of the JVM running as the coordinator and to determine whether the JVM heap size needs to be increased or more coordinators are needed.

- ▶ Do not put the coordinator in a JVM that is under a constant heavy load

A coordinator needs CPU cycles available to react quickly to core group events, for example, when one of the core group servers crashes, the coordinator needs to update the status of that server and to communicate the event to all group members. Any heavily loaded system will prevent the HAManager from functioning properly and thus jeopardize your WebSphere high availability environment. A heavily loaded system is not one that is running at 100% of its capacity, but one that is running a much heavier workload than it can handle. Under such load, a system will be unable to schedule the tasks required for a coordinator in a reasonable amount of time. For example, if an alarm was scheduled to run every two seconds, such a system would have the alarm firing but not being processed for maybe 20 seconds. Clearly, this severely disrupts the normal processing of the HA services. The HAManager displays a warning message (HMGR0152) when it detects these types of scheduling delays. If this message is observed then you need to take action to prevent the circumstances causing this problem. That may include stopping paging, retuning the thread pools to a more reasonable level for the number of CPUs in the system or using more or faster hardware for the application. The usual causes for this problem are either:

- Swapping
- The server is configured to use a very large number of threads for when compared with the number of CPUs on the system.
- Other processes or JVMs on the system are causing thread scheduling delays (that is, the total number of active busy threads on the system is too high for it).

These symptoms typically impact everything on the system.

- ▶ Use a JVM that is not often started or stopped

When a preferred coordinator server is stopped then a small amount of CPU will be used on all machines in the core group to recover the coordinator state. This typically takes well under a second. If the server is later restarted then the same process is repeated as the new coordinator recovers the state.

Using multiple coordinators can reduce the rebuild time by spreading the rebuild and steady state coordinator CPU/memory load over multiple computers. However, the amount is CPU required in steady state is practically zero and the rebuild CPU is also minimal in almost all scenarios. The only scenarios where the rebuild times would increase beyond subsecond times are when there is a very large number of JVMs in the core group as well as a very large number of clustered applications or JMS destinations. Very large means tens of thousands.

To explain the election of coordinators, let's look at an example configuration with a core group consisting of the following JVMs in lexical order:

- app1Node/Ejb1
- app1Node/Web1
- app1Node/nodeagent
- app2Node/Ejb2a
- app2Node/Ejb2b
- app2Node/Web2a
- app2Node/Web2b
- app2Node/nodeagent
- dmNode/dmgr

Possible configurations of preferred coordinator servers and election results are:

Table 9-1 Example configurations and results of preferred coordinator servers

Number of coordinators	Preferred coordinator servers	Inactive group members	Elected coordinators
1	nil	nil	app1Node/Ejb1
1	app1Node/Ejb1	app1Node/Ejb1	app1Node/Web1
2	app1Node/Ejb1	nil	app1Node/Ejb1, app1Node/Web1
2	dmNode/dmgr, app1Node/Ejb1	nil	dmNode/dmgr, app1Node/Ejb1
2	app1Node/Ejb1, dmNode/dmgr	nil	app1Node/Ejb1, dmNode/dmgr
2	app1Node/Ejb1, app1Node/Web1	app1Node/Ejb1	app1Node/Web1, app1Node/nodeagent

As JVMs take up the role of a coordinator during a view change, a message is written to the SystemOut.log file as seen in Example 9-1.

Example 9-1 Message for a JVM becoming an active coordinator

```
[10/27/04 17:24:46:234 EDT] 00000013 CoordinatorIm I    HMGR0206I: The
Coordinator is an Active Coordinator for core group DefaultCoreGroup.
```

For JVMs that do not become coordinators, the message in Example 9-2 is displayed.

Example 9-2 Message for a JVM joining a view not as a coordinator

```
[10/27/04 18:49:03:016 EDT] 0000000a CoordinatorIm I    HMGR0228I: The
Coordinator is not an Active Coordinator for core group DefaultCoreGroup.
```

The preferred coordinator list can be dynamically changed. If there is a newly elected coordinator, a message as in Example 9-3 on page 471 is written to the SystemOut.log.

Example 9-3 Message for an active coordinator retiring from the role

```
[10/27/04 17:28:51:766 EDT] 00000013 CoordinatorIm I    HMGR0207I: The
Coordinator was previously an Active Coordinator for core group
DefaultCoreGroup but has lost leadership.
```

Coordinator failure

When a JVM process with the active coordinator is no longer active (because it is stopped or crashes), the HAManager elects the first inactive server in the preferred coordinator servers list. If there is none available, it will simply elect the lexically lowest named inactive server. If there are fewer JVMs running than the number of coordinators in the core group settings, then all running JVMs are used as coordinators.

The newly elected coordinator initiates a state rebuild, sending a message to all JVMs in the core group to report their states. This is the most CPU-intensive operation of a coordinator. Multicast is the ideal transport type for this operation. See “Multicast” on page 478 for more information.

How many coordinators do I need?

Most medium-scale core groups only need one coordinator. The following are possible reasons for increasing the number of coordinators:

- Heavy heap usage found in the verbose garbage collection log of the JVM acting as the active coordinator.

- ▶ High CPU usage when a newly elected coordinator becomes active.

Both of these conditions are only a problem under the following circumstances:

- ▶ There are thousands of WebSphere clusters deployed in the core group.
- ▶ There are thousands of JMS destinations deployed in the core group.
- ▶ A WebSphere Extended Deployment application using partitioning is having more than 5000 partitions.

For 99% of customers, it won't be necessary to use more than a single coordinator. Normally, you'll use a preferred server to pin the coordinator to a server that doesn't start/stop typically but if that server fails then the coordinator will move to the lexically lowest JVM.

9.2.2 Transport buffer

The underlying message transport of HAManager is a reliable publish/subscribe messaging service, known as Distribution and Consistency Services (DCS). A buffer is created to hold unprocessed incoming messages and outgoing messages that have not been acknowledged.

The default memory size is 10MB with the rationale to reduce memory footprint. As the buffer is shared with DRS for HTTP session replication and stateful session bean state replication, you may wish to increase the buffer size if your WebSphere environment has high replication demands. When an application server is running low on the transport buffer, a HMGR0503I message is displayed in the SystemOut.log of the JVM logs. This is not an error message. It is simply an informational message that indicates a congestion event has occurred in the transport buffer. The messages that are not sent during the congestion are retried later or they will be sent as a batch to make use of the transport buffer more efficiently.

Congestion should normally only occur when doing a lot of session replication or when a large core group is started by simultaneously starting all members. Congestion can be reduced by tuning the buffer size.

The ideal setting is very dependant on load but during some internal benchmarks, we used buffer sizes of 80MB for a cluster that was processing 20,000 HTTP requests per second and each request resulted in 10k of session state to replicate. This is an extreme that very few customers would see in practice but gives an idea of the scale of things.

Important: The replication function was heavily tuned for the 6.0.2 release of WebSphere and we recommend at least that level to customers with heavy HTTP session replication needs.

Example 9-4 Informational message when the transport buffer is running low

HMGR0503I: The message path of the internal data stack named <data stack name> is highly congested.

To change the transport buffer size, open the WebSphere Administrative Console and click **Servers -> Application server -> <AppServer_Name> -> Core group service** (under Additional Properties). See Figure 9-3 on page 473 for the configuration panel.

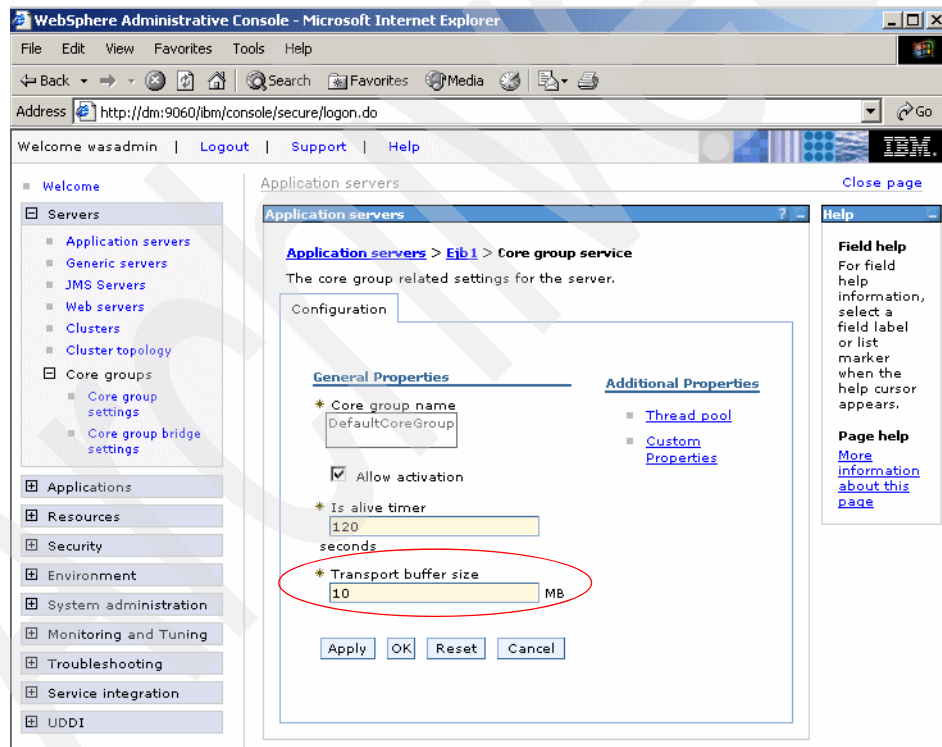


Figure 9-3 Change the transport buffer size of an application server

Important: This setting is per application server. You need to perform this change for all the application servers in the core group.

9.2.3 Distribution and Consistency Services

Distribution and Consistency Services (DCS) provide the underlying group services framework for the HAManager such that each application server process knows the health and status of JVMs and singleton services. It basically provides view synchronous services to the HAManager. DCS itself uses RMM as its reliable pub/sub message framework. RMM is an ultra high speed publish/subscribe system that WebSphere uses internally for its core group communication fabric as well as for DRS traffic.

9.2.4 Core group policy

A core group policy determines how many and which members of a *high availability group* are activated to accept work at any point of time. Each service or group can have its own policy. A single policy manages a set of high availability groups (HA groups) using a matching algorithm. A high availability group must be managed by exactly one policy. For more information about HA groups please refer to 9.3, “High availability group” on page 479.

Policies can be added, deleted or edited while the core group is running. These changes take effect immediately. There is no need to restart any JVMs in the core group for a policy change to take effect.

To create or edit a core group policy, click **Servers -> Core groups -> Core group settings -> <core_group_name> -> Policies -> New** or **<existing_policy>**. The panel in Figure 9-4 is shown.

The screenshot shows a web-based configuration interface for 'Core groups'. The breadcrumb navigation is 'Core groups > DefaultCoreGroup > Policies > Clustered TM Policy'. Below the navigation, there is a description: 'Define a policy that will activate one group member in the core group.' The main configuration area is titled 'Configuration' and contains two panels: 'General Properties' and 'Additional Properties'.

General Properties:

- Name:** Clustered TM Policy
- Policy type:** One of N policy
- Description:** TM One-Of-N Policy
- Is alive timer:** 120 seconds
- Quorum:** ☐
- Fail back:** ☒
- Preferred servers only:** ☐

Additional Properties:

- [Custom properties](#)
- [Match criteria](#)
- [Preferred servers](#)

At the bottom of the configuration area are four buttons: 'Apply', 'OK', 'Reset', and 'Cancel'.

Figure 9-4 Editing or creating a core group policy

There are five types of core group policies available:

- ▶ All active policy
- ▶ M of N policy
- ▶ No operation policy
- ▶ One of N policy
- ▶ Static policy

We describe the more common policies in detail below.

One of N Policy

Only one server activates the singleton service at a time under this policy. If a failure occurs, the HAManager will start the service on another server. Make sure that all external resources are available to all high availability group members at all times when using this policy. For example, if database access is required for messaging, all members should have the remote database catalogued. If there are transaction logs, they should be put on a Network Access Storage (NAS) that is available to all members. This is the recommended policy for systems that require automatic failovers and do not use external high availability software.

The one-of-N policy has the following additional options to cater for different usage scenarios:

- ▶ Preferred servers

You can specify an ordered list of servers that the HAManager observes when choosing where to run a singleton service.

- ▶ Quorum

Leave this checkbox unchecked.

This option is only needed for WebSphere Extended Deployment customers using partitioning *and* using hardware to enforce quorums. If you are however using WebSphere Partition Facility and the appropriate hardware, then please contact IBM support to help you configuring this setting correctly.

- ▶ Fail back

When enabled, a singleton service will be moved to a preferred server when one becomes available. One example usage is the Transaction Manager. When the failing server with the Transaction Manager becomes online again, it should re-acquire the Transaction Manager service as Transaction Manager failover only caters for recovery processing.

- ▶ Preferred servers only

This option makes a singleton service to run exclusively on servers in the preferred servers list.

Two default one-of-N policies are defined for the DefaultCoreGroup: Clustered TM Policy for the high availability of a Transaction Manager and Default SIBus Policy for protecting the Service Integration Bus (messaging) services.

Important: The default policies should never be edited, changed or deleted. They can be overridden by new policies that have a more specific matchset.

No operation policy

Using this policy, the HAManager never activates a singleton service on its own. It is primarily intended to be used with an external clustering software, such as the IBM HACMP. The software controls where to activate a singleton service by invoking operations on the HAManager MBean.

Typically, this mode is used when overall system infrastructure dictates the singleton service to have dependencies on resources managed outside WebSphere. For example, Transaction Manager logs may be placed on a Journaled File System (JFS) that is present on a SAN disk. Only one server can mount a JFS file system at a time, even though it is on a shared disk.

Recovery time is significantly reduced than the cold standby model in previous versions of WebSphere Application Server since all JVMs are running before a failover event. The expensive JVM startup time is avoided during the critical failover time.

Static

This policy should be used when you want the singleton service to run on a specific high availability group member. If the member is not online, the singleton service will not be running. The singleton service will not automatically failover. Manual intervention is required. The fixed member can be changed on the fly without restarting WebSphere. This option is useful when failover is undesirable. If the server then fails, the service can be moved to another server by updating the server name on the policy and saving it.

9.2.5 Match criteria

Every singleton service is managed by a high availability group to which a policy is assigned at runtime. The assignment is done by comparing the match criteria of the set of available policies against the properties of the high availability groups. The policy with the strongest match will be assigned to the HA group. You can edit a policy by clicking **Servers -> Core groups -> Core group settings -> New** or **<existing_core_group_name> -> Policies -> New** or **<existing_policy_name> -> Match Criteria**.

Refer to Table 9-2 on page 477 and Table 9-3 on page 477 for match criteria name/value pairs of messaging engines and Transaction Managers:

Table 9-2 Match criteria for messaging engines

Name	Value	Match targets
type	WSAF_SIB	All messaging engines
WSAF_SIB_MESSAGING_ENGINE	Name of your messaging engine	One particular messaging engine
WSAF_SIB_BUS	Name of your bus	All messaging engines in a bus
IBM_hc	Name of your cluster	All messaging engines in a cluster

Table 9-3 Match criteria for Transaction Managers

Name	Value	Match targets
type	WAS_TRANSACTIONS	All Transaction Managers

Name	Value	Match targets
IBM_hc	Cluster name	All Transaction Managers in a cluster
GN_PS	Home server name	One particular Transaction Manager

9.2.6 Transport type

A transport type is the type of network communication a core group uses to communicate to its members. There are three types of transports available:

- ▶ Multicast
- ▶ Unicast
- ▶ Channel Framework

No matter which transport type is used, there is always a socket between each pair of JVMs for point-to-point messages and failure detection. For example, if you have a total of eight JVMs in your core group, then every JVM will have seven sockets to others.

Multicast

Multicast is a high performance protocol and the HAManager is designed to perform best in the multicast mode especially when using very large core groups. Typical scenarios for multicast are:

- ▶ There are over 70 JVMs in the core group
- ▶ Many JVMs are created on a small number of large SMPs

Publishing a message in this mode is efficient as the publish only has to transmit once. Only a fixed number of threads is used independent of the number of JVMs in a core group. However, consider the following factors to decide if multicast is suitable for your environment:

- ▶ Multicast typically requires all JVMs in the core group to be on the same subnet (TTL can be tuned, please contact IBM support for details).
- ▶ All members in a core group receive multicast messages. JVMs waste CPU cycles on discarding messages that are not intended for them.

Unicast

Communications between JVMs are performed via the direct TCP sockets between each pair of JVMs under this transport mode. The unicast mode has the following advantages and disadvantages:

- ▶ Unicast is WAN friendly. There is no limitation to localize JVMs on a single subnet.

- ▶ Publishing a message which is intended only for a small number of servers is more effective than multicast. Servers that have no interest in the message will not waste CPU cycles on discarding messages.
- ▶ Only a fixed number of threads are used regardless of the number of JVMs.
- ▶ Publishing a message to a large number of servers is more expensive considering each message is sent once per destination JVM.

Channel Framework

This default transport mode has similar pros and cons as unicast. It is more flexible than unicast in the sense that a core group in this transport is associated with a channel chain, and a chain can use HTTP tunneling, SSL or HTTPS. The performance of channel framework is around 50% less than unicast for tasks such as HTTP session replication. It is a trade-off option between performance and flexibility for different environments. SSL and HTTP tunneling are only available using the channel framework transport.

If the transport type in a core group is changed, all JVMs in that group must be restarted. The following is the recommended procedure for changing the transport type:

1. Stop all cluster members and Node Agents in the core group.
2. Modify the transport type using the WebSphere Administrative Console by clicking **Servers -> Core groups -> Core group settings -> <core_group_name>**.
3. Change the Transport type setting on the Configuration tab.
4. Perform a manual synchronization using the command line utility **syncNode**.
5. Start all Node Agents.
6. Start all cluster members and application servers.

9.3 High availability group

High availability groups are dynamic components created from a core group. Each group represents a highly available singleton service. The active members in a group are ready to host the service at any time. See Figure 9-5.

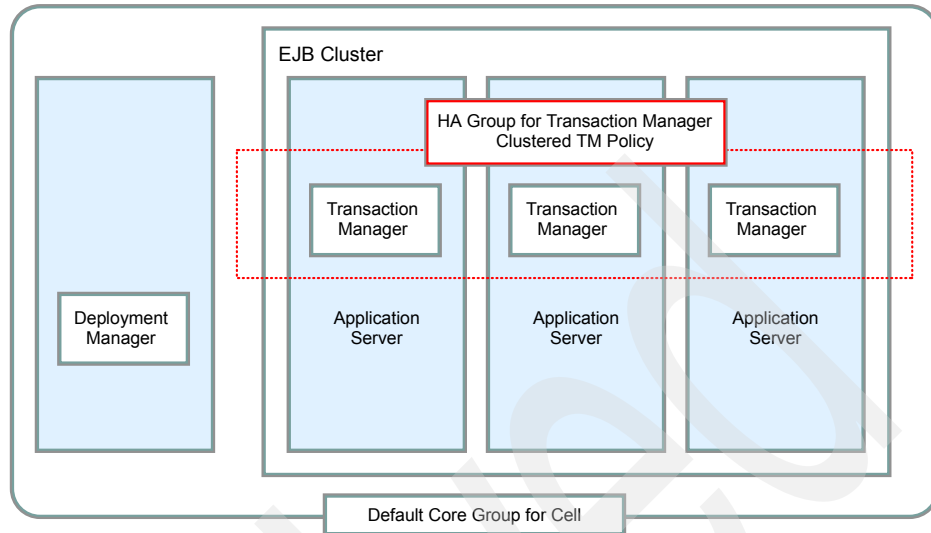


Figure 9-5 High availability group for a Transaction Manager

To view a list of high availability groups, click **Servers -> Core groups -> Core group settings -> <core_group_name>**. Then select the **Runtime** tab. Specify a match criterion for a list of specific high availability group(s) or an asterisk (*) as a wildcard to get a complete list of groups. For example, specifying **type=WAS_TRANSACTIONS** results in a list of Transaction Manager high availability groups. See Figure 9-6 on page 480.

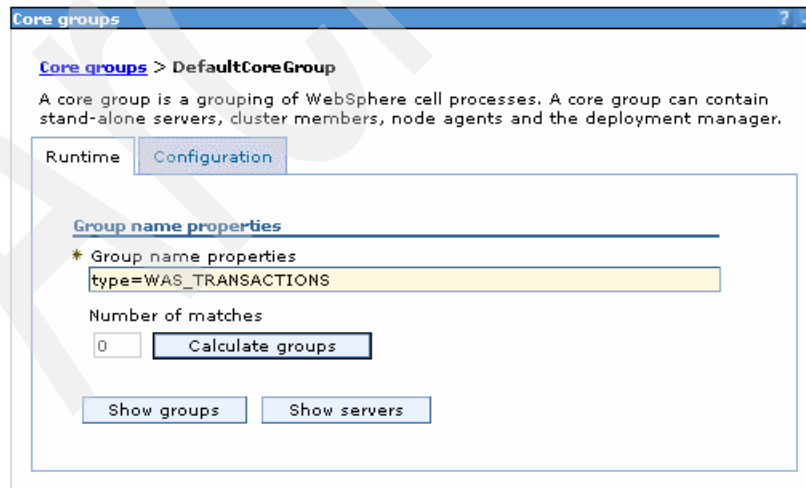


Figure 9-6 Find high availability groups for Transaction Managers

When you click **Show groups**, a list of high availability groups that match the criteria you specified is displayed as shown in Figure 9-7 on page 481. Each high availability group is displayed along with its associated policy.

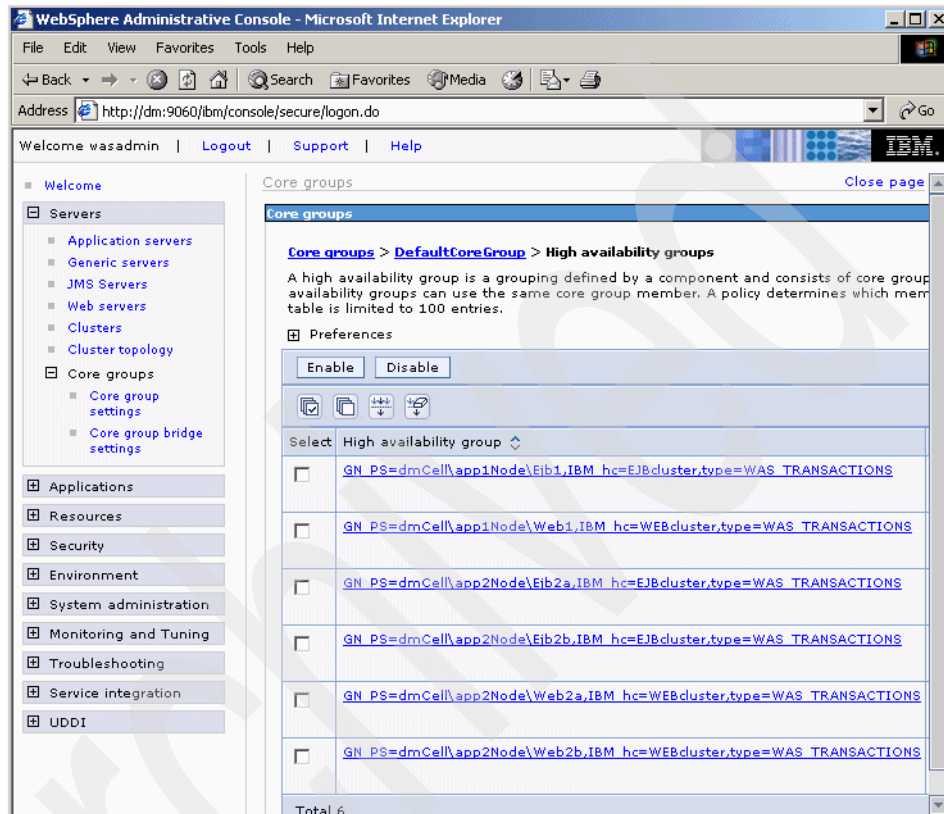


Figure 9-7 Listing high availability groups of Transaction Managers

Select any high availability group and a list of group members is displayed as shown in Figure 9-8 on page 482. Only running JVMs are displayed in the group member list. From here you can manage the group members by activating, deactivating, enabling or disabling them.

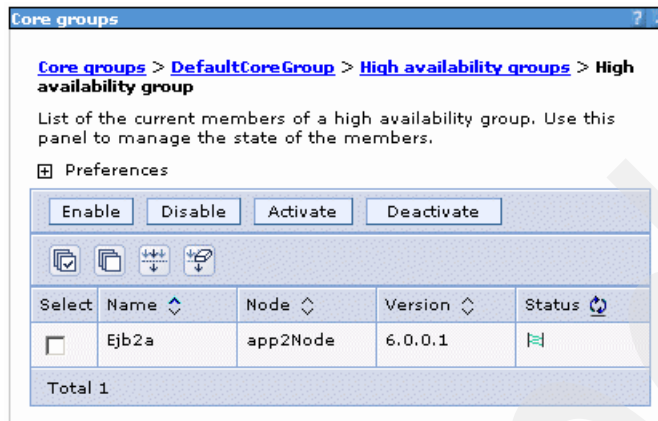


Figure 9-8 Managing a high availability group

9.4 Discovery of core group members

A JVM that is starting in a core group goes through three stages before joining the group:

1. Not connected

The JVM has not established network connectivity with other group members. It will send a single announcement message if the multicast transport mode is used. Or it will send a message to each member of the group if unicast is used. It sends multiple messages in unicast because it doesn't know which other members are started.

2. Connected

The JVM has already opened a stream to all current members of the installed view. The coordinator will consider this JVM as a candidate to join the view. A view is the set of online JVMs ready for running singleton services.

3. In a view

The JVM is a full participant in a core group at this stage. The view is updated and installed in all members.

When a new view is installed, message HMGR0218I is displayed in the SystemOut.log file of each JVM in the group indicating how many JVMs are currently a member of the view.

Example 9-5 Message HMGR0218I for a new view being installed

```
[10/27/04 18:52:20:781 EDT] 00000011 CoordinatorIm I    HMGR0218I: A new core  
group view has been installed. The view identifier is  
(16:0.dmCell\app1Node\Ejb1). The number of members in the new view is 5.
```

JVMs in the current view constantly try to discover others that are not in the view. Each in-view JVM periodically tries to open sockets to JVMs that are not in the current view. This process continues until all JVMs in the core group are in the view.

Attention: When running a large number of JVMs on a single box, the `FIN_WAIT` parameter of the operating system may need to be tuned down to prevent running out of ephemeral ports. Please consult your OS documentation for details.

9.5 Failure Detection

The HAManager monitors JVMs of core group members and initiates failover when necessary. It uses two methods to detect a process failure:

- ▶ Active failure detection
- ▶ TCP KEEP_ALIVE

9.5.1 Active failure detection

A JVM is marked as failed if its heartbeat signals to its core group peers are lost for a specified interval. The DCS sends heartbeats between every JVM pair in a view. With the default settings, heartbeats are sent every 10 seconds and 20 heartbeat signals must be lost before a JVM is raised as a suspect and a failover is initiated. The default failure detection time is therefore 200 seconds.

This setting is very high and should be modified by most customers in a production environment. A setting of 10-30 seconds is normally recommended for a well tuned cell.

Note: Settings as low as 6 seconds are possible but typically only used with WebSphere Extended Deployment WPF applications. Please contact IBM support or services if there is a need to tune these settings below the recommended range.

When a JVM failure is detected, it is “suspected” by others in the view. This can be seen in the SystemOut.log shown in Example 9-6 on page 494. The new view

installation in this case is fast in order to achieve fast recovery. New view installations are slower for new views generated from JVM startups. Otherwise, there would be frequent view installations when several JVMs are started together.

Heartbeat delivery can be delayed due to a number of commonly-seen system problems:

- **Swapping**

When a system is swapping, the JVM could get paged and heartbeat signals are not sent or received in time.

- **Thread scheduling thrashing**

Java is not a real time environment. When there are a lot of runnable threads accumulated in a system, each thread will suffer a long delay before getting scheduled. Threads of a JVM may not get scheduled to process heartbeat signals in a timely fashion. This thread scheduling problem also impacts the applications on that system as their response times will also be unacceptable. Therefore, systems must be tuned to avoid CPU starving or heavy paging.

Any of the above problems can cause instability in your high availability environment. After tuning the system not to suffer from swapping or thread thrashing, the heartbeat interval can be lowered to increase the sensitivity of failure detection.

Use the core group custom properties listed in Table 9-4 to change the heartbeat frequency:

Table 9-4 Changing the frequency of active failure detection

Name	Description	Default value
IBM_CS_FD_PERIOD_SECS	This is the interval between heartbeats in seconds.	10
IBM_CS_FD_CONSECUTIVE_MISSED	This is the number of missed heartbeats to mark a server as a suspect.	20

Heartbeating is always enabled regardless of the message transport type for the HAManager.

9.5.2 TCP KEEP_ALIVE

If a socket between two peers is closed then the side receiving the closed socket exception will signal its peers that the other JVM is to be regarded as failed. This

means that if a JVM panics or exits then the failure is detected as quickly as the TCP implementation allows. If the failure is because of a power failure or a network failure then the socket will be closed after the period defined by the KEEP_ALIVE interval of the operating system. This is normally a long time and should be tuned to more realistic values in any WebSphere system. A long KEEP_ALIVE interval can cause many undesirable behaviors in a highly available WebSphere environment when systems fail (including database systems).

This failure detection method is however less prone to CPU/memory starvation from swapping or thrashing. Both failure detectors together offer a very reliable mechanism of failure detection.

Attention: The TCP KEEP_ALIVE value is a network setting of your operating system. Changing its value may have side-effects to other processes running in your system. Refer to “Connection Timeout setting” on page 335 for details on how to overwrite this value in a WebSphere environment.

9.6 JMS high availability

WebSphere Application Server V6 includes a pure Java implementation of JMS messaging engines. Given each application server has database connectivity to the database server for persistent messages, the messaging service can be protected by a one-of-N policy. For details on JMS high availability and configuration options, please refer to Chapter 12, “Using and optimizing the default messaging provider” on page 643.

9.7 Transaction Manager high availability

The WebSphere transaction service writes to its transaction logs when WebSphere handles global transactions that involve two or more resources. Transaction logs are stored on disk and are used for recovering in-flight transactions from system crashes or process failures. As the transactions are recovered, any database locks related to the transactions will be released. If the recovery is somehow delayed, not only the transactions cannot be completed, database access may be impaired due to unreleased locks.

As depicted in Figure 9-9 on page 486, as two application servers perform two-phase commit (2PC) transactions, they place table/row locks in the database, depending on the configuration of lock granularity.

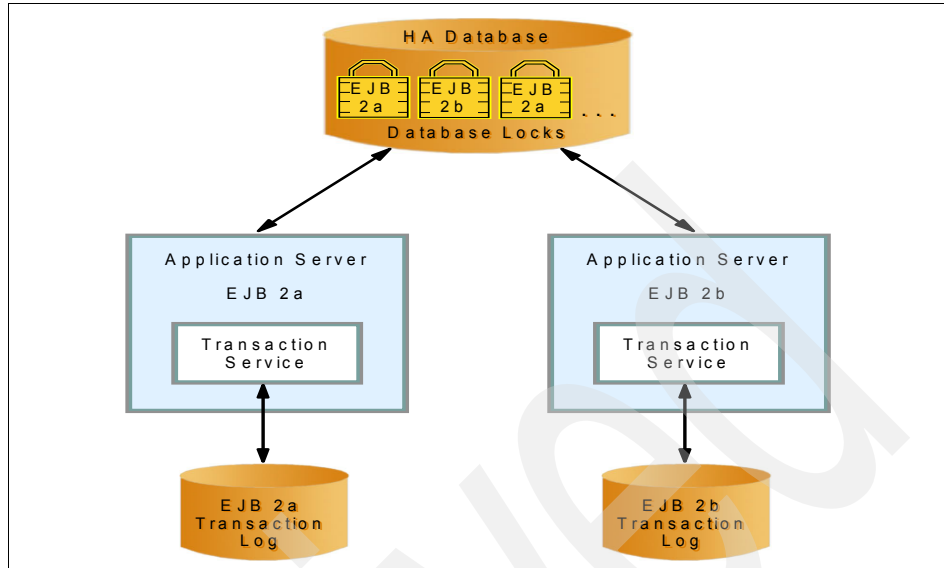


Figure 9-9 Two phase commit transactions and database locking

In the event of a server failure, the transaction service of the failed application server is out of service. Also, the in-flight transactions that have not be committed may leave locks in the database, which will block the surviving server from gaining access to locked records.

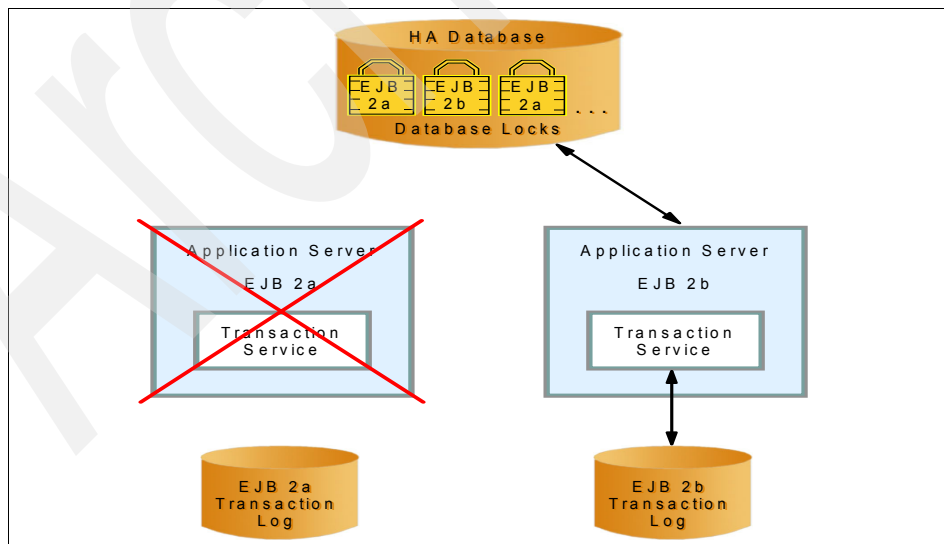


Figure 9-10 Server failure during an in-flight transaction

The only way to complete the transactions and release the locks is to restart the failed server, or start the application server process in another box that has access to the transaction logs.

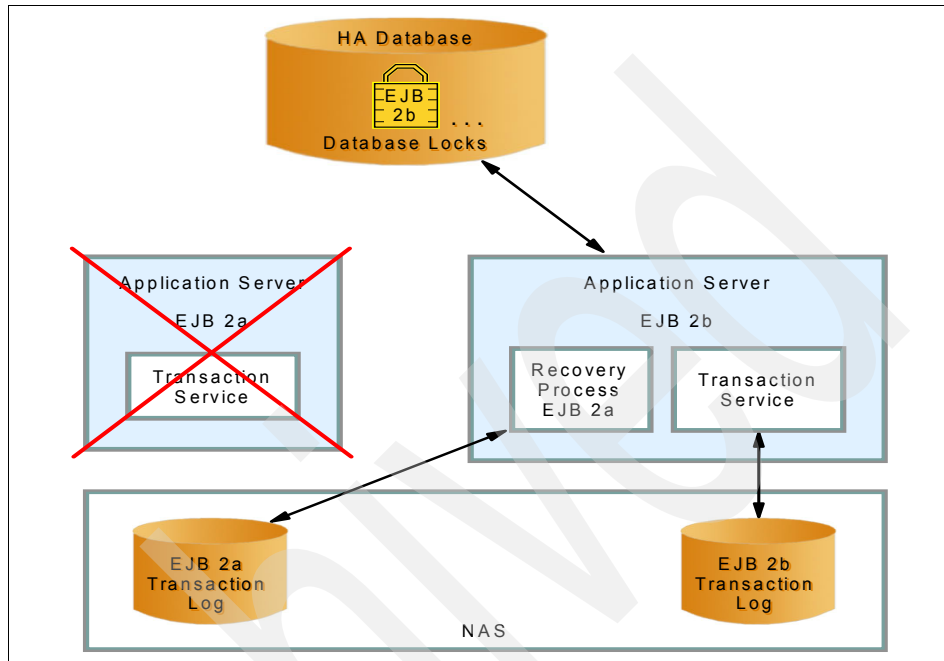


Figure 9-11 Recovery of failed transactions

Using the new HAManager, a recovery process will be started in other members of the cluster. The in-flight transactions are committed and locks released. We describe this configuration in detail in the next sections.

9.7.1 Transaction Manager HA of previous versions of WebSphere

In previous versions of WebSphere Application Server, transaction log recovery could only be achieved by restarting an application server which leads to slow recovery time from a failure. This is known as a cold failover as the backup server needs to start an application server process during a failover. WebSphere also requires IP failover for transaction log recovery in older versions of WebSphere. Figure 9-12 on page 488 depicts a typical HA setup with previous versions of WebSphere Application Server:

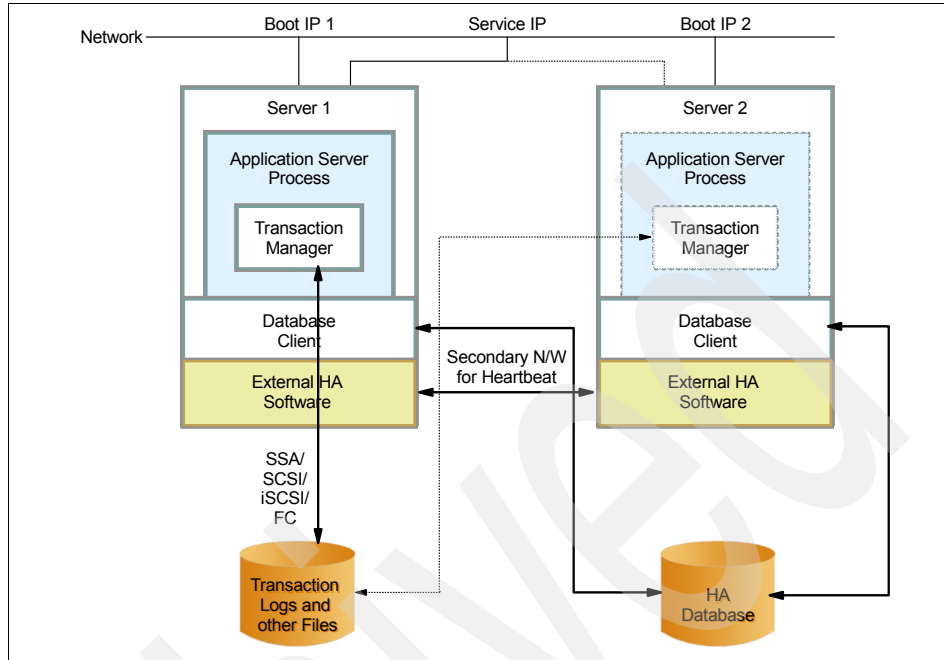


Figure 9-12 Traditional Transaction Manager high availability setup

A shared drive is attached to both servers using SSA/SAN fabric. It holds the configuration repository, log files, transaction logs and the WebSphere Application Server binaries as well. Both servers have their own IP addresses, and share a virtual IP address through which clients can access the application server. An external HA software, such as IBM HACMP or Tivoli System Automation, is used to manage the resource group of the virtual IP address, shared disk and its file systems, and scripts to start/stop the WebSphere Application Server process.

If the active server crashes or fails, the HA software moves the resource group to the backup server. It involves assigning and mounting the shared drive on the backup server, assigning the virtual IP address and then starting the WebSphere Application Server process. Although this is a proven solution, it has a number of disadvantages:

- ▶ Recovery time is slow. The application server process can only be started during a failover to recover the transaction logs and resolve any in-doubt transactions. This can potentially take more than five minutes due to JVM start times.
- ▶ There is a single virtual IP address to be failed over, which leads to a limitation of having both servers on the same subnet. This is not desirable

when you want to have your application servers physically located at different sites for a high level of resilience.

- The configuration of this HA solution is highly complex. Additional HA software is necessary. And there are timing and dependency issues in starting components in the resource group.

However, there are scenarios where an external solution is still best. For example, if you want to keep a messaging engine collocated with a DB2 database and HACMP is being used to manage the database availability. In this case, HACMP must also be used to tell the HAManager on which machine to run the messaging engine.

By leveraging latest storage technologies, WebSphere Application Server V6 offers a much simpler HA configuration. The newly introduced peer-to-peer hot-failover model allows transaction recovery to be performed in a much shorter time. While the new version can still work along side with external HA software, IBM WebSphere Application Server Network Deployment V6 itself can be a Transaction Manager HA solution with the right environment. Refer to 9.7.2, “Hot-failover of Transaction Manager using shared file system” on page 489 for details.

9.7.2 Hot-failover of Transaction Manager using shared file system

This is the simplest of all Transaction Manager HA setups. It requires all cluster members to have access to a shared file system on NAS or SAN, where the transaction logs are stored, see Figure 9-13 on page 490.

Normally, every cluster member runs its own Transaction Manager. When a failover occurs, another cluster member will be nominated to perform recovery processing for the failed peer according to the Clustered TM Policy of the core group. The recovery process completes in-doubt transactions, releases any locks in the back-end database and then releases the transaction logs. No new work will be performed beyond recovery processing. The Transaction Manager will fail back when the failed server is restarted.

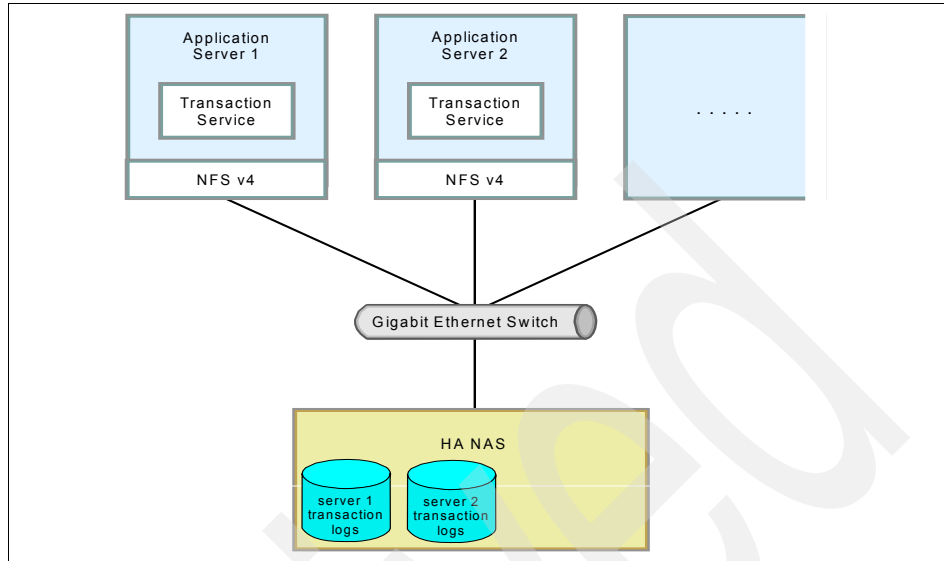


Figure 9-13 Network Deployment V6 transaction service high availability using NAS

Environment

The shared file system must support automatic lock recovery in order to make the peer-to-peer (one-of-N) recovery model work. File system locking is obviously vital to prevent corruption of the transaction log files. Lock recovery is necessary to ensure peer cluster members can access the transaction logs once held by the failed member.

Important: All shared file systems that are compatible with WebSphere 6.0 currently use lock leasing. The lock lease times should be tuned to an acceptable time. Most default lock lease times are around the 45 second mark.

While the HAManager can be tuned to fail over in 10 seconds, this won't help if the lock lease time is 45 seconds as the locks won't free up until 45 seconds. We recommend lock lease times of 10 seconds and setting the HAManager failure detection time to just over this or 12 seconds (2 second heart beats, 6 missed heartbeats means suspect).

The following file system types are supported when the one-of-N policy is used:

- ▶ Network File System (NFS) version 4
- ▶ Windows Common Internet File System (CIFS)
- ▶ IBM TotalStorage® SAN File System

Basically, any shared filesystem with the following characteristics should work:

- ▶ Flush means all changes to the file are written to persistence store (physical disks or NVRAM on a SAN server).
- ▶ File locks use a leasing mechanism to allow locks held by failed machines to be released in a reasonable amount of time without requiring the failed server to restart.

Please note that although a shared filesystem that complies to the above characteristics should work, IBM only supports configurations that it has tested as working (see the list of supported file systems above).

Restriction: NFS version 3 or earlier is not supported as a shared file system when the one-of-N policy is used.

Use the static or no operation policy if your system does not support NFS version 4 or CIFS.

AIX 5.2 supports NFS version 3 while AIX 5.3 supports NFS version 4.

For more information about IBM NAS and SAN technologies, please go to IBM TotalStorage homepage at

<http://www.storage.ibm.com>

The Network Attached Storage (NAS) and all the cluster members are connected together in a network. There is no limitation on placing servers on the same subnet as long as they can make connections to each other.

All cluster members must be running WebSphere Application Server V6 or later. Peer recovery processing does not work with V5 cluster members.

Configuration

Follow these steps to make the Transaction Manager highly available:

1. Install WebSphere Application Server V6 on all nodes. It is not necessary to have the WebSphere Application Server binaries on the shared file system.
2. Create a cluster and add cluster members (on the different nodes). Refer to Chapter 8, "Implementing the sample topology" on page 387 for details.
3. Enable the cluster to enforce high availability for transaction logs.

High availability for persistent service is a cluster setting. Click **Servers -> Clusters -> <cluster_name>**. Select the **Enable high availability for persistent services** checkbox on the Configuration tab shown in Figure 9-14 on page 492.

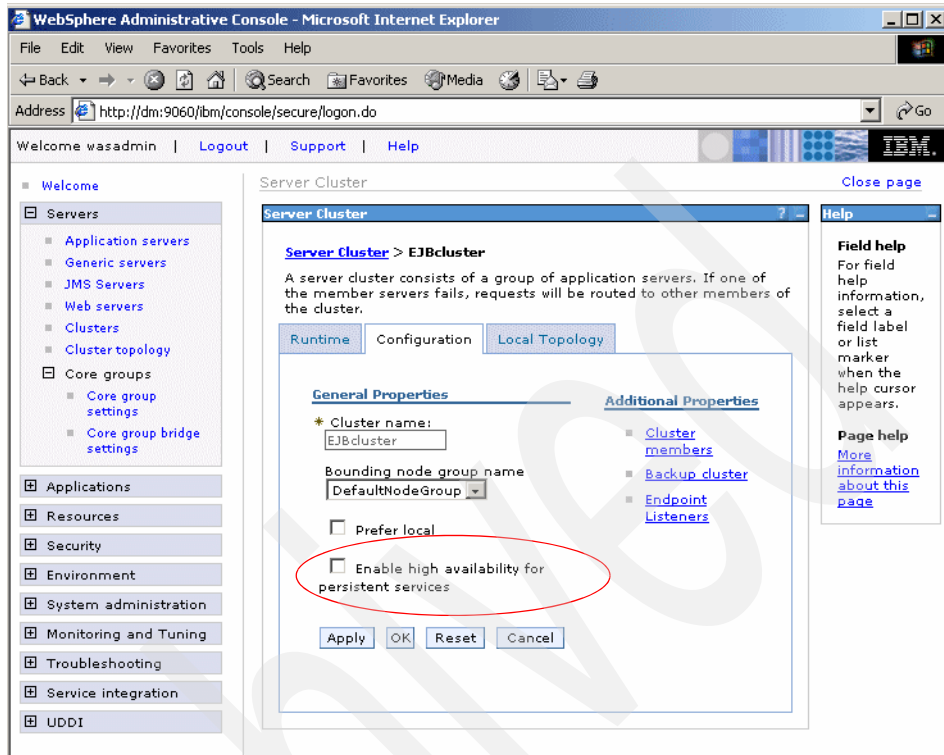


Figure 9-14 Enable high availability for the transaction service of a cluster

4. Stop the cluster.
5. Change the transaction log directories of all cluster members.

Click **Servers -> Application servers -> <AppServer_Name> -> Container Services -> Transaction Service**. Enter the transaction logs location on the NFS mount point of the cluster member into the Transaction log directory field.

Tips:

- It is recommended that you use the hard option in the NFS mount command **mount -o hard** to avoid data corruption.
- We recommend the same settings that a database would use if it used the shared file system. Most NAS vendors document recommended settings for this environment. Settings that work for a database such as DB2 or Oracle will also work for WebSphere 6.0 transaction logs.

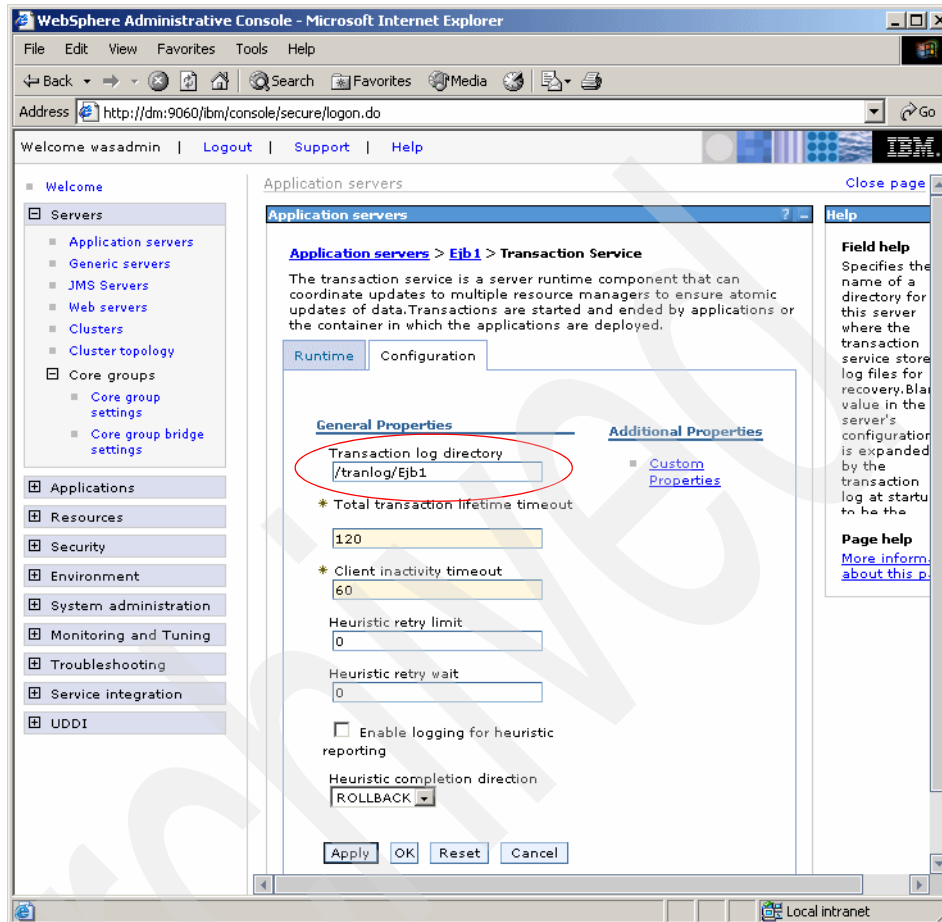


Figure 9-15 Change the transaction log directory of a cluster member

6. Copy the existing transaction logs to the shared file system. Make sure the location and file permissions are correct.
7. Save and synchronize the configuration.
8. Start the cluster.

The Transaction Manager is highly available after the cluster restart. To verify how many high availability groups are backing up the singleton service, follow the steps in 9.3, “High availability group” on page 479.

Failover test

Now we perform a simple simulation of a JVM crash of application server Ejb2a, and server Ejb2b performs the peer recovery of transaction logs for Ejb2a.

When looking at the log in Example 9-6 you will notice an application server named Ejb1 which is an application server in our sample topology. Although it has been disabled in our HA setup, it is still used in the view name as it has the lexically lowest name in the cluster.

Important: We have disabled the Ejb1 server during these tests only because of the specifics of our setup. Our Ejb1 server runs on Windows while Ejb2a and Ejb2b run on AIX. In our sample topology, we only have a shared file system on AIX, which cannot be accessed by the Windows server. Therefore, we could not include Ejb1 for the Transaction Manager tests.

It is important that you understand that application servers on Windows and AIX *can* indeed coexist in a transaction manager HA scenario if all systems of the environment have access to the shared disk. But, the principal issue is that the directory names are very different between Windows and UNIX. A UNIX machine will not understand a Windows file name and vice versa. Clever use of WebSphere environment variables can be used to allow this but it's an unusual scenario.

Both application servers are up and running. We issue the AIX command `kill -9 <pid_of_Ejb2a>` to the application server process to terminate the process. The Ejb2b application server immediately detects the event from the closed socket connection and raises Ejb2a as a suspect. A new view is immediately installed to reflect the change. Then a recovery process is started in the Ejb2b JVM to recover the transaction logs of Ejb2a. The recovery process is complete around three seconds after the server is killed. This is a big improvement when compared with the minutes required by previous versions of WebSphere. Please see Example 9-6 for the details.

Example 9-6 SystemOut.log of Ejb2b after terminating Ejb2a

```
[11/8/04 15:34:26:489 EST] 00000017 RmmPtpGroup W DCSV1111W: DCS Stack
DefaultCoreGroup at Member dmCell\app2Node\Ejb2b: Suspected another member
because the outgoing connection from the other member was closed. Suspected
members is dmCell\app2Node\Ejb2a. DCS logical channel is View|Ptp.
[11/8/04 15:34:26:547 EST] 00000017 DiscoveryRmmP W DCSV1111W: DCS Stack
DefaultCoreGroup at Member dmCell\app2Node\Ejb2b: Suspected another member
because the outgoing connection from the other member was closed. Suspected
members is dmCell\app2Node\Ejb2a. DCS logical channel is Connected|Ptp.
[11/8/04 15:34:26:694 EST] 00000017 RmmPtpGroup W DCSV1111W: DCS Stack
DefaultCoreGroup.DynamicCache at Member dmCell\app2Node\Ejb2b: Suspected
another member because the outgoing connection from the other member was
```


closed. Suspected members is dmCell\app2Node\Ejb2a. DCS logical channel is View|Ptp.

[11/8/04 15:34:26:886 EST] 00000017 DiscoveryRmmP W DCSV1111W: DCS Stack DefaultCoreGroup.DynamicCache at Member dmCell\app2Node\Ejb2b: Suspected another member because the outgoing connection from the other member was closed. Suspected members is dmCell\app2Node\Ejb2a. DCS logical channel is Connected|Ptp.

[11/8/04 15:34:27:344 EST] 00000017 DiscoveryServ W DCSV1111W: DCS Stack DefaultCoreGroup at Member dmCell\app2Node\Ejb2b: Suspected another member because the outgoing connection from the other member was closed. Suspected members is dmCell\app2Node\Ejb2a. DCS logical channel is Discovery|Ptp.

[11/8/04 15:34:27:831 EST] 00000017 VSync I DCSV2004I: DCS Stack DefaultCoreGroup at Member dmCell\app2Node\Ejb2b: The synchronization procedure completed successfully. The View Identifier is (139:0.dmCell\app1Node\Ejb1). The internal details are [0 0 0 0 0 0 0 0].

[11/8/04 15:34:28:151 EST] 00000017 VSync I DCSV2004I: DCS Stack DefaultCoreGroup.DynamicCache at Member dmCell\app2Node\Ejb2b: The synchronization procedure completed successfully. The View Identifier is (110:0.dmCell\app1Node\Ejb1). The internal details are [0 0 0 0 0 0].

[11/8/04 15:34:28:300 EST] 00000019 CoordinatorIm I HMGR0228I: The Coordinator is not an Active Coordinator for core group DefaultCoreGroup.

[11/8/04 15:34:28:425 EST] 00000019 CoordinatorIm I HMGR0218I: A new core group view has been installed. The view identifier is (140:0.dmCell\app1Node\Ejb1). The number of members in the new view is 8.

[11/8/04 15:34:28:448 EST] 00000019 CoreGroupMemb I DCSV8050I: DCS Stack DefaultCoreGroup at Member dmCell\app2Node\Ejb2b: New view installed, identifier (140:0.dmCell\app1Node\Ejb1), view size is 8 (AV=8, CD=8, CN=8, DF=11)

[11/8/04 15:34:28:701 EST] 00000017 ViewReceiver I DCSV1033I: DCS Stack DefaultCoreGroup at Member dmCell\app2Node\Ejb2b: Confirmed all new view members in view identifier (140:0.dmCell\app1Node\Ejb1). View channel type is View|Ptp.

[11/8/04 15:34:29:297 EST] 00000049 RecoveryDirec A WTRN0100E: Performing recovery processing for a peer WebSphere server (FileFailureScope: dmCell\app2Node\Ejb2a [-1788920684])

[11/8/04 15:34:29:324 EST] 00000049 RecoveryDirec A WTRN0100E: All persistant services have been directed to perform recovery processing for a peer WebSphere server (FileFailureScope: dmCell\app2Node\Ejb2a [-1788920684])

[11/8/04 15:34:29:601 EST] 00000049 RecoveryDirec A WTRN0100E: All persistant services have been directed to perform recovery processing for a peer WebSphere server (FileFailureScope: dmCell\app2Node\Ejb2a [-1788920684])

[11/8/04 15:34:29:771 EST] 0000004a RecoveryManag A WTRN0028I: Transaction service recovering 0 transactions.

[11/8/04 15:34:29:668 EST] 0000001c DataStackMemb I DCSV8050I: DCS Stack DefaultCoreGroup.DynamicCache at Member dmCell\app2Node\Ejb2b: New view installed, identifier (111:0.dmCell\app1Node\Ejb1), view size is 5 (AV=5, CD=5, CN=5, DF=6)

```
[11/8/04 15:34:29:895 EST] 0000001c RoleMember I DCSV8052I: DCS Stack
DefaultCoreGroup.DynamicCache at Member dmCell\app2Node\Ejb2b: Defined set
changed. Removed: [dmCell\app2Node\Ejb2a].
[11/8/04 15:34:31:500 EST] 00000019 HAManagerImpl I HMGR0123I: A GroupUpdate
message was received for a group that does not exist. The group name is
drs_agent_id=Ejb2a\baseCache\001663\1,drs_inst_id=1100098001663,drs_inst_name=b
aseCache,drs_mode=0,policy=DefaultN00PPolicy.
[11/8/04 15:34:31:561 EST] 00000017 ViewReceiver I DCSV1033I: DCS Stack
DefaultCoreGroup.DynamicCache at Member dmCell\app2Node\Ejb2b: Confirmed all
new view members in view identifier (111:0.dmCell\app1Node\Ejb1). View channel
type is View|Ptp.
.....
```

Tip: In AIX, use the command `kill -9 <pid_of_JVM_process>` to simulate a JVM crash. The `kill -9 <pid>` command issues a SIGKILL signal to immediately terminate a process, while the `kill <pid>` command by default sends a SIGTERM signal to normally stop a process.

9.7.3 Hot-failover of transaction logs using external HA software

Some customers may wish to use file systems that only allow one box to mount at a time, for example Journal File System (JFS) of AIX, to store WebSphere transaction logs. External clustering software is necessary to make such a WebSphere solution highly available. The external HA software will perform detection and controls the allocation of shared resources like shared file systems and IP addresses.

Unlike in previous versions, the WebSphere application server process will not be started using the HA software. Instead, the process starts as a daemon at boot time and restarts upon crashes. The HA software will coordinate the dependency of shared file systems and the Transaction Manager. The proper startup procedures would be:

1. Mount the file system for transaction logs
2. Activate the Transaction Manager

Configuration

Follow the configuration steps in “Configuration” on page 491. Make sure the shared file systems are mounted on their respective hosts before copying the transaction logs.

1. Stop the cluster.

2. Create a policy to override the default Clustered TM Policy for Transaction Managers. Click **Servers -> Core groups -> Core group settings -> <core_group_name> -> Policies -> New**.
 - a. From the drop-down list, select **No operation policy** and click **Next**. See Figure 9-16.

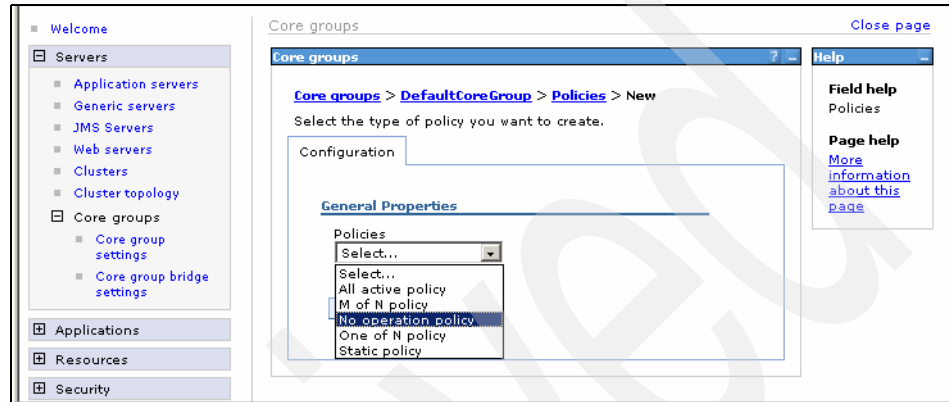


Figure 9-16 Create a No operation policy for Transaction Managers - select type

- b. Fill in the name and description as depicted in Figure 9-17 on page 498, then click **OK**. A warning regarding match criteria is issued, therefore we define the match criteria next.

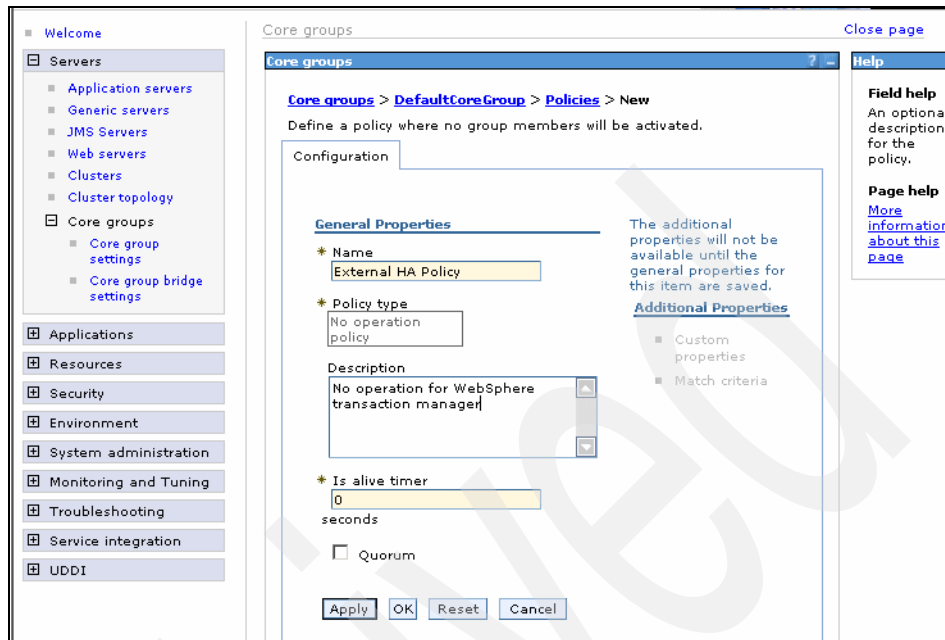


Figure 9-17 Create a No operation policy for Transaction Managers - define properties

3. Click **Match criteria** -> **New** to create a match set for Transaction Managers in a cluster. Enter **IBM_hc** as the Name and your **cluster name** as the Value. Click **OK**. See Figure 9-18.

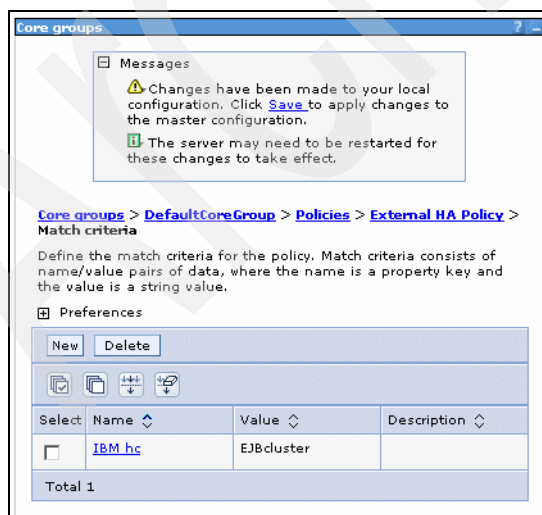


Figure 9-18 Creating a match set for Transaction Managers in the EJBcluster

4. Save and synchronize all nodes.
5. Instead of starting the cluster using the WebSphere Administrative Console, start the application servers as a daemon or service. Notice that the application server will pause to wait until the Transaction Manager is ready before displaying the ready for e-business message. It will continue to hang until the Transaction Manager is started by the external HA software.

That is all that needs to be done on the WebSphere side. Now you need to configure your HA software and set up resource groups. There should be one resource group for each Transaction Manager. Every resource group has its associated physical disks and their file systems, and scripts to start/stop an application. In our case, we need scripts to activate/deactivate Transaction Manager high availability group members via the HAManager JMX interface. This can be done using a wsadmin script or a Java admin client which calls the HAManager MBean to activate/deactivate a particular high availability group member. Step-by-step details of HA software configurations are outside the scope of this book.

Dynamic caching

Server-side caching techniques have long been used to improve Internet performance of Web applications. In general, caching improves response time and reduces system load. Until recently, caching has been limited to *static content*, which is content that rarely changes. However, performance improvements are greatest when *dynamic content* is also cached. Dynamic content is content that changes frequently, or data that is personalized. Caching dynamic content requires proactive and effective invalidation mechanisms, such as event-based invalidation, to ensure the freshness of the content. Implementing a cost effective caching technology for dynamic content is essential for the scalability of today's dynamic, data-intensive, e-business infrastructures.

This chapter covers the following topics:

- ▶ "Introduction" on page 502
- ▶ "Using WebSphere dynamic cache service" on page 515
- ▶ "WebSphere dynamic caching scenarios" on page 528
- ▶ "WebSphere external caching scenarios" on page 558
- ▶ "Conclusion" on page 609
- ▶ "Benchmarking Trade 3" on page 611

10.1 Introduction

Performance became a term frequently used during every WebSphere Application Server project and caching became a key technology to reduce cost and improve the response time.

This chapter introduces the dynamic cache service provided by WebSphere Application Server V6. We explain how and when caching can be used to improve the performance of WebSphere Application Server solutions.

Important: What we are describing in this chapter is the administrative aspect of configuring dynamic caching which happens during application deployment. However, caching is something that needs to be taken into account during application architecture, design and development to be really effective. An application design that does not incorporate caching needs is often difficult to configure caching policies for. This chapter does not go into detail on the design and architecture of the application to make it suitable for caching.

In most cases, enabling caching is a task performed during application deployment, though in some cases the application developers need to perform specific customization, which are required by the caching framework. For example, when it comes to more granular caching techniques, such as command caching, application developers have to develop the code according to WebSphere caching specifications. We address such customization in “Command caching” on page 540.

10.1.1 WWW caching services

First, there are two basic types of content from the caching point of view:

- Static content

Static content does not change over long periods of time. The content type can be HTML, JSP rendering less dynamic data, GIF, JPG, etc.

Note: JSP rendering less dynamic data is considered to be static content from the caching point of view. When JSP is rendered, static HTML code is generated and this is cached.

Static content is characterized by content that is not generated on the fly and has fairly predictable expiration values. Such content typically resides on the file system of the Web server or application server and is served unchanged to the client. To a large extent such content can be declared to be cacheable and have expiration values associated with it. From the perspective of the

Proxy server, the standard HTTP headers associated with such content describe how long this content is valid, Furthermore, any intermediary and certainly the client is free to cache such content locally.

In general, when it comes to caching, only public content is cached at any of the intermediaries unless the intermediary performs authentication and authorization of content. WebSphere Application Server on the other hand, can cache private content because any authentication and authorization of the resource happens prior to fetching content from the cache.

► Dynamic content

Dynamic content includes frequently updated content (such as exchange rates), as well as personalized and customized content. Although it is changing content, at the same time it must be stable over a long enough time for meaningful reuse to occur. However, if some content is very frequently accessed, such as requests for pricing information of a popular stock, then even a short time of stability may be long enough to benefit from caching.

Secondly, it is important to understand that caching goes beyond an applications' own infrastructure. For example, there are many cache instances currently deployed on the Internet. Figure 10-1 outlines the different tiers of caching for Web applications.

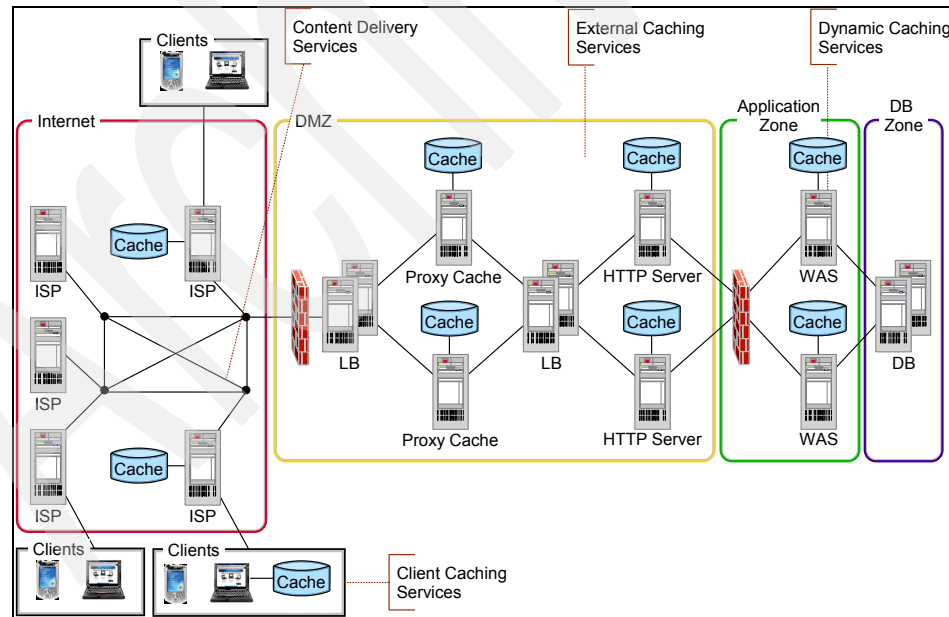


Figure 10-1 Content caching on the internet

The content caching tiers are:

- Client caching services

These services can be provided by any kind of Web browsers. Web browsers can store received content locally on the client device. Unfortunately, very often you will experience that interpretations of HTTP cache directives are browser dependent.

- Internet content delivery services

These services provide the replication and caching infrastructure on the Internet. The infrastructure is based on standard concepts, protocols and taxonomy. The main actors are Internet Services Providers (ISP) and Content Delivery Network (CDN) service providers, such as Akamai (<http://www.akamai.com>), or Speedera (<http://www.speedera.com>).

The physical infrastructure is built using cache appliances such as Network Appliance (<http://www.netapp.com>), Blue Coat (<http://www.bluecoat.com>), or Cisco Cache Engine (<http://www.cisco.com>).

These services can also support the dynamic composition of page fragments in the network using a standard called Edge Side Includes (ESI, <http://www.esi.org>). “WebSphere external caching scenarios” on page 558 explains how WebSphere Application Server V6 supports ESI.

- External caching services

These services provide front-end caching of both static and dynamic pages. The technical component can be the Caching Proxy, part of WebSphere Edge Components, IBM HTTP Server, or the Web server plug-in. All of them can intercept requests from a client, retrieve the requested information from content-hosting machines, and deliver that information back to the client. In addition, they can store cacheable content in a local cache (memory and/or disk) before delivering it to the requestor. This enables the external caches to satisfy subsequent requests for the same content by delivering it directly from the local cache, which is much quicker than retrieving it again from the content host.

These services provide both whole page and fragment caching; and again the dynamic composition of page fragments is based on ESI.

We further discuss external caching services in “WebSphere external caching scenarios” on page 558.

- Dynamic caching services

These services provide the caching of servlets, JSPs, commands, or Web services. They work within an application servers’ Java Virtual Machine (JVM) by intercepting calls to cacheable objects.

“WebSphere dynamic caching scenarios” on page 528 focuses on these dynamic caching services.

This chapter is focused on the dynamic caching services and their integration with external cache services.

In most cases only a combination of dynamic and external services can power high volume Web sites to achieve the required level of scalability and performance. When implementing such a combination, the impact on data integrity and accuracy needs to be also considered. For example, if you have a WebSphere Commerce implementation and you use WebSphere Edge Components to cache product pages, it can take some time until the WebSphere Edge Components cache is synchronized with updates to the product database used by WebSphere Commerce. This time can be as long as the time-to-live parameter set for the product pages.

The question to be asked is, how can you make sure that all these different caching services will support your specific application? The answer to this question is to use standard protocols, such as HTTP/1.1 and ESI/1.0.

Traditional HTTP caching is mostly used in cases where the whole page is cacheable and does not contain personalized content. On the other hand, ESI allows caching to operate on the level of fragments rather than whole pages.

HTTP/1.1 provides basic cache mechanisms in the form of implicit directives to caches. It also allows the use of explicit directives for the HTTP cache called “Cache - Control”. The Cache-Control header allows a client or server to transmit a variety of directives in either requests or responses. These directives typically override the default caching algorithms.

Detailed information about the HTTP/1.1 caching directives are included in RFC 2616 (<http://www.faqs.org/rfcs/rfc2616.html>). Additionally, RFC 3143 lists known HTTP Proxy Caching problems (<http://www.faqs.org/rfcs/rfc3143.html>).

Note: An additional level of caching, which is not described in this chapter, is database caching. There are a number of products available for database caching, for example DB2 DBCache, TimesTen, Oracle 9i IAS - Relational Cache, Versant's enJin.

For more information about database caching, refer to this Web site:

<http://www.almaden.ibm.com/u/mohan/>

Here you will find a link to the presentation "Caching Technologies for Web Applications."

10.1.2 Fragment caching

Most dynamic Web pages consist of multiple smaller and simpler page fragments. Some fragments are static (such as headers and footers), while others are dynamic (such as fragments containing stock quotes or sport scores). Breaking a page into fragments or components makes effective caching possible for any page, even a highly dynamic page.

These are the main goals of fragment caching:

- ▶ Achieve benefits of caching for personalized pages.
- ▶ Reduce cache storage requirements, by sharing common fragments among multiple pages which helps maximize fragment reusability.
- ▶ Move cache into the network to multiply above benefits.

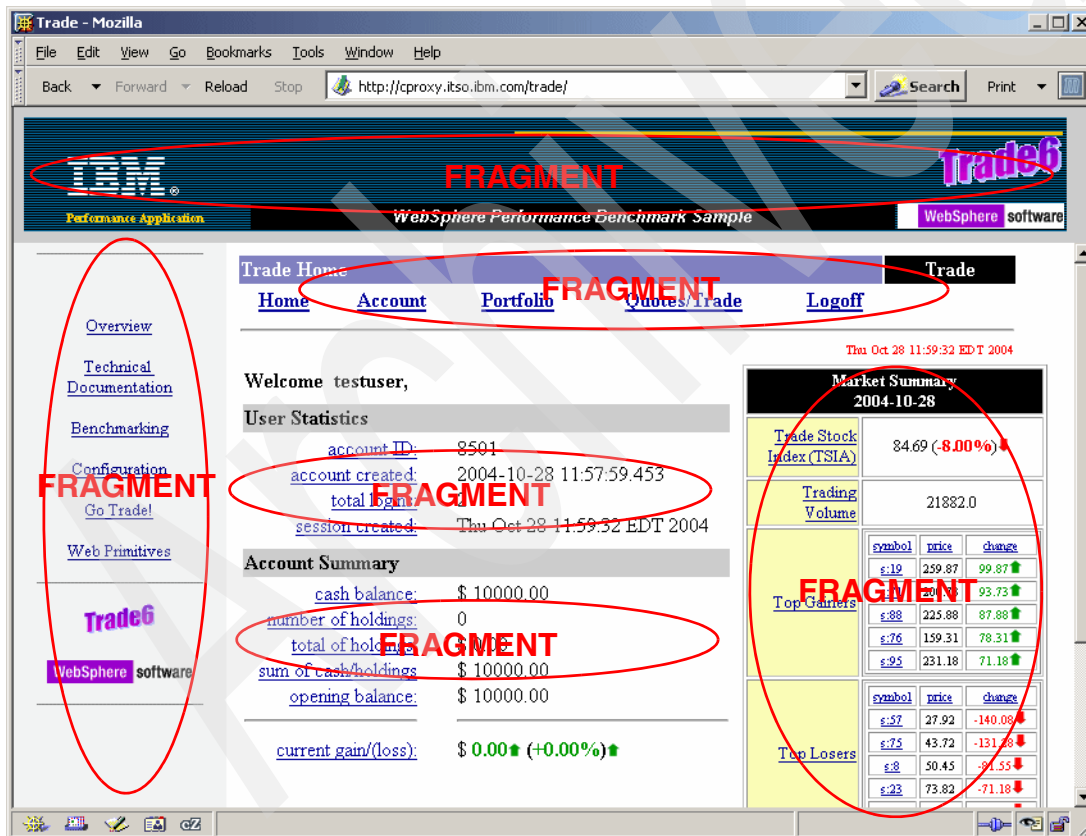


Figure 10-2 Page fragments

Figure 10-2 on page 506 shows that when a page is broken down into fragments based on reusability and cacheability, some or all of the fragments (for example, headers, footers, and navigation bars for all users; market summary for user groups) may become reusable and cacheable for a larger audience. Only fragments that are not cacheable need to be fetched from the back-end, thereby reducing server-side workload and improving performance. Even a very short time of caching can improve performance, for example if you can cache the Market Summary information just for 10 seconds, it can make a big difference during peak hours.

Web pages should be fragmented to cache dynamic content effectively within the enterprise infrastructure and at the content distribution network. However, in some cases, even caching the most granular, final formatted fragment is not sufficient. Under such circumstances, caching at the raw data level is the next granular technique that can be used (see 10.5.3, “Command caching” on page 540).

Web page design also plays an important role in determining where dynamic data is cached. One example is personalized pages. Although personalized, these pages contain user specific, nonuser-specific, locale sensitive, secure, non security sensitive dynamic data. To maximize the benefit of caching dynamic content, these types of pages should be fragmented as finely as possible. They can be cached independently at different locations. For example, the nonuser-specific, non security sensitive fragments or components are generally useful to many users, and thus can be cached in a more public space and closer to the users. The security sensitive data should be cached behind the enterprise firewall, yet as close to the edge of the enterprise as possible.

10.1.3 Dynamic caching scenarios

The key issue with caching dynamic content is to determine what should be cached, where caching should take place, and how to invalidate cached data.

In a multi-tier e-business environment, the WebSphere Dynamic Cache service can be activated at the business logic and/or presentation layer. It can also control external caches on servers, such as WebSphere Caching Proxy or IBM HTTP Server. When external caching is enabled, the cache matches pages with their universal resource identifiers (URIs) and exports matching pages to the external cache. The contents can then be served from the external cache instead of the application server, which saves resources and improves performance.

To simulate real production systems, we explain the following scenarios:

- ▶ WebSphere Dynamic Cache
 - At the presentation logic layer (servlet/JSP resulting caching)

- At the business logic layer (command caching)
- WebSphere external caching
 - Using Web server plug-in with ESI
 - Using IBM HTTP Server with FRCA
 - Using WebSphere Caching Proxy

Caching of Web services and Web services client caching is covered in Chapter 24, “Web services caching” of the redbook *WebSphere Version 6 Web Services Handbook Development and Deployment*, SG24-6461.

We do not focus directly on the caching techniques used by Content Delivery Networks, but we explain the IBM Web server plug-in, which has ESI surrogate capabilities.

Please refer to Figure 10-3 for a graphical representation of these options.

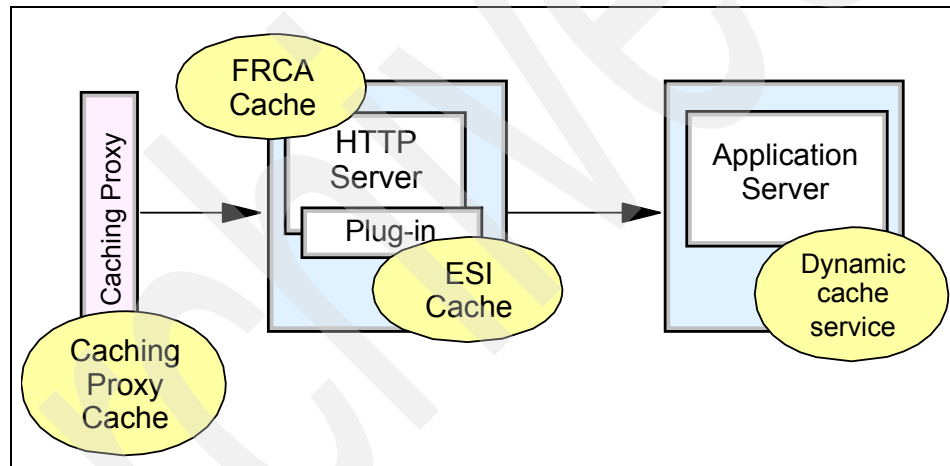


Figure 10-3 Dynamic caching options

When you design the caching solution for your application, you may want to combine different caching techniques. For example you may use a combination of dynamic caching and Web server plug-in caching, or a dynamic caching and WebSphere Caching Proxy combination. This chapter provides you with information that helps you to design your individual caching solution.

We are using the redbook’s WebSphere Application Server infrastructure, which is described in detail in Chapter 8, “Implementing the sample topology” on page 387, for our scenarios. We use the WebSphere Performance Benchmark sample application Trade 6, because apart from other significant features, Trade 6 supports key WebSphere components such as dynamic caching.

As you can see in Figure 10-4 we can use caching services at different tiers of the application infrastructure. Our scenarios explain the appropriate caching techniques for the different tiers.

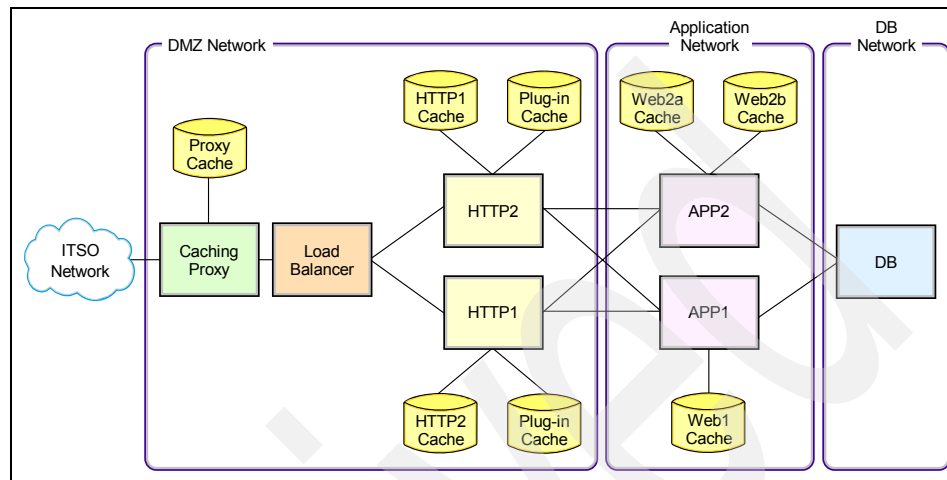


Figure 10-4 Redbook caching infrastructure

10.2 What is new in WebSphere V6 dynamic caching

IBM WebSphere Application Server V6 includes some new features, enhancements and modifications in the dynamic cache service. These are:

10.2.1 Dynamic Content Provider interface

A cacheable servlet or JSP file might contain a state in the response that does not belong to the fragment for that servlet or JSP. When the state changes, the cached servlet or JSP is not valid for caching.

Dynamic Content Provider (DCP) allows the developer to use its interface and add user exits to the servlet response object.

When the servlet response is generated, DCP is called to provide the output of the dynamic content of that fragment.

10.2.2 Cache instances

Cache instances provide additional locations where the dynamic cache service can store, retrieve and share data.

Each cache instance is independent from each other, and it is not affected by other cache instances.

Applications running on an application server can access cache instances on other application servers as long as they are part of the same replication domain.

WebSphere Application Server V5.1 Dynamic Cache provided a feature called cache instance. In V6, this feature was extended and now it provides two types of cache instances: *servlet cache instance* and *object cache instance*.

The servlet cache instance can store servlets, JSPs, Struts, Tiles, command objects and SOAP requests. It allows applications like WebSphere Portal Server and WebSphere Commerce to store data in separate caches.

The object cache instance is used to store, distribute and share Java objects. The following APIs are provided so the applications can interact with the object cache instances:

- ▶ DistributedObjectCache
- ▶ DistributedMap

The DistributedMap and DistributedObjectCache interfaces are simple interfaces for the dynamic cache. Using these interfaces, J2EE applications and system components can cache and share Java objects by storing a reference to the object in the cache. The default dynamic cache instance is created if the dynamic cache service is enabled in the Administrative Console. This default instance is bound to the global Java Naming and Directory Interface (JNDI) namespace using the name `services/cache/distributedmap`.

Tip: If you are not using command caching yet but want to start using it, then it is recommended that you use the DistributedMap API rather than command caching as described in 10.5.3, “Command caching” on page 540.

For more information about DistributedMap and DistributedObjectCache please refer to the WebSphere InfoCenter at:

http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tdyn_distmap.html

10.2.3 Caching Struts and Tiles applications

In WebSphere V6, the servlet and JSP caching was enhanced to allow for easy Struts and Tiles caching.

Struts is an open source framework for building Web applications that are based on the model-view-controller (MVC) architecture. The Struts framework provides

its own controller component and integrates with other technologies to provide the model and the view. The primary focus of this framework is to provide a control layer for the Web application, reducing both construction time and maintenance costs.

The Tiles framework builds on the `jsp:include` feature and comes bundled with the Struts Web application framework. This framework helps to reduce the duplication between JavaServer Pages (JSP) files as well as make layouts flexible and easy to maintain. The Tiles structure provides a full-featured, robust framework for assembling presentation pages from component parts.

For more information about Struts and Tiles caching, please refer to 10.5.2, “Struts and Tiles caching” on page 537.

10.2.4 Cache replication

Cache replication allows cached objects to be shared across multiple servers in a cluster.

WebSphere Application Server V5.1 already provided this feature, but all instances shared the same settings. In WebSphere Application Server V6 the replication settings are configured per instance. For more information refer to 10.2.4, “Cache replication” on page 511.

10.3 The `cachespec.xml` configuration file

WebSphere dynamic caching service policies are defined in the *cachespec.xml* file. This file defines, for example, which objects to cache (the `<cache-entry>`) and where (for example in the dynamic cache service of the application server itself or in an external cache). Dependencies between objects can also be defined as well as timeouts and invalidation.

The cache parses the `cachespec.xml` file when the server starts and also dynamically when the file is changed, and extracts a set of configuration parameters from each `<cache-entry>` element. The `<cache-entry>` elements can be inside the root `<cache>` element or inside a `<cache-instance>` element. Cache entries that are in the `<root>` element are cached with the default cache instance. Cache entries that are in the `<cache-instance>` element are cached in that particular cache instance. Different cacheable objects have different `<class>` elements. You can define the specific object a cache policy refers to using the `<name>` element.

Location

It is recommended that you store the cachespec.xml file with the deployment module. You could also place a global cachespec.xml file in the application server properties directory.

The cachespec.dtd file is available in the application server properties directory. The cachespec.dtd file defines the legal structure and the elements that can be in your cachespec.xml file.

10.3.1 cachespec.xml elements

This section highlights only some important cachespec.xml elements. For a complete list and more detailed information please refer to the WebSphere InfoCenter article "cachespec.xml file" at

http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/rdyn_cachespec.html

<cache>

The root element of the cachespec.xml file is <cache> and contains <cache-instance> and <cache-entry> elements. The <cache-entry> elements can also be placed inside of <cache-instance> elements to make that cache entry part of a cache instance other than the default cache instance.

<cache-instance>

The name attribute is the Java Naming and Directory Interface (JNDI) name of the cache instance that is set in the Administrative Console or using wsadmin while creating/editing a cache-instance.

<cache-entry>

Each cache entry must specify certain basic information that the dynamic cache uses to process that entry. The <cache-entry> elements in cachespec.xml can include the following elements: class, name, sharing-policy, property, and cache-id.

<class>

This element is required and specifies how the application server interprets the remaining cache policy definition. The value servlet refers to servlets and JavaServer Pages (JSP) files that are deployed in the WebSphere Application Server servlet engine. The value command refers to classes using the WebSphere command programming model. The value static means files that contain static content. The webservice class extends the servlet with special component types

for Web services requests. The value `JAXRPCClient` is used to define a cache entry for the Web services client cache.

Here are some examples of the usage of the class element:

- ▶ `<class>command</class>`
- ▶ `<class>servlet</class>`
- ▶ `<class>webservice</class>`

<name>

Specifies a cacheable object. Some guidelines for the name element are:

- ▶ For servlets and JSP files as well as for static files, if the `cachespec.xml` file is stored with the application, this element can be relative to the specific Web application context root. If `cachespec.xml` is stored in the application servers's properties directory, then the full URI of the JSP file, servlet, or static file must be specified.

For a Web application with a context root, the cache policy for files using the static class must be specified in the Web application, and not in the properties directory. As mentioned earlier, the recommended location is in the Web application and not in the properties directory.

- ▶ For commands, this element must include the package name, if any, and class name, including a trailing `.class` of the configured object.

You can specify multiple `<name>` elements within a `<cache-entry>` if you have different mappings that refer to the same servlet. Here are some examples on how to use the `<name>` element:

- ▶ `<name>com.mycompany.MyCommand.class</name>`
- ▶ `<name>default_host:/servlet/snoop</name>`
- ▶ `<name>com.mycompany.beans.MyJavaBean</name>`
- ▶ `<name>mywebapp/myjsp.jsp</name>`
- ▶ `<name>mywebapp/myLogo.gif</name>`

<sharing-policy>

This is an important setting when working in a clustered environment and using cache replication. The value of this element determines the sharing characteristics of cache entries. The default value is `not-shared` and is the value assumed when the `<sharing-policy>` element is not specified. Possible values are:

- ▶ `not-shared`
- ▶ `shared-push`
- ▶ `shared-pull`
- ▶ `shared-push-pull`

Usage example: `<sharing-policy>shared-push-pull</sharing-policy>`.

For single server environments, the only valid value is not-shared.

For more information about cache replication and the meaning of these values please see “Cache replication” on page 547.

<property>

You can set optional properties on a cacheable object, such as a description of the configured servlet. The class determines valid properties of the cache entry.

Usage example: `<property name="key">value</property>`

where key is the name of the property for this cache entry element, and value is the corresponding value. Important properties (keys) for our sample scenario and the tests in this chapter are:

- ▶ **EdgeCacheable**

This value is valid for servlet/JSP caching and can be set to true or false. The default is false. If the property is true, then the servlet or JSP file is externally requested from an Edge Side Includes processor. Whether or not the servlet or JSP file is cacheable depends on the rest of the cache specification.

- ▶ **ExternalCache**

The value specifies the external cache name. The external cache name needs to match the external cache group name. This key is only valid for servlet/JSP caching. See 10.6, “WebSphere external caching scenarios” on page 558 for more information.

<cache-id>

To cache an object, the application server must know how to generate a unique ID for different invocations of that object. These IDs are built either from user-written custom Java code or from rules defined in the cache policy of each cache entry. Each cache entry can have multiple cache ID rules that are executed in order until either:

- ▶ A rule returns a non-empty cache ID, or
- ▶ No more rules are left to execute

If none of the cache ID generation rules produce a valid cache ID, the object is not cached.

Each `<cache-id>` element defines a rule for caching an object and is composed of the sub-elements component, timeout, inactivity, priority, property, idgenerator, and metadatagenerator.

10.3.2 Dynamic Cache Policy Editor

The Dynamic Cache Policy Editor is a tool that helps creating and editing the dynamic cache policies file `cachespec.xml`. This tool validates cache policies against its XML schema, provides assistance in completing content and restricts changes that are invalid during runtime. It also includes a tool that analyses servlets and JSPs, generates the cache policies for them, and adds those policies to the cache policy file.

The Dynamic Cache Policy Editor is provided as an Eclipse plug-in and plugs into WebSphere Studio Application Developer V5.1.0 and higher or Application Server Toolkit (AST) V5.1.0 and higher.

It can be downloaded from:

<http://www.alphaworks.ibm.com/tech/cachepolicyeditor>

Restriction: At the time of writing this redbook, the Dynamic Cache Policy Editor could not be used with Rational Application Developer V6 or AST V6.

As it is not expected that the GUI changes much once available for the V6 tools, we decided to include this section to show the functionality of the tool. All screenshots and descriptions are based on the available version plugged into AST V5.1.

Please check the download site for new versions of the product.

Please refer to 10.7, “Using the Dynamic Cache Policy Editor” on page 586 for details on how to install and use the Cache Policy Editor.

The next sections explain some `cachespec.xml` entries needed for the various caching scenarios explained in this chapter. You can either create these entries manually using any editor or refer to 10.7.3, “Examples: Creating `cachespec.xml` entries with the Dynamic Cache Policy Editor” on page 592 where we discuss how to create some of those examples using the Dynamic Cache Policy Editor.

10.4 Using WebSphere dynamic cache service

Before we dive into the different scenarios, we explain the following general configuration tasks:

- ▶ Installing Dynamic Cache Monitor
- ▶ Enabling and configuring WebSphere dynamic cache service

10.4.1 Installing Dynamic Cache Monitor

The Dynamic Cache Monitor (Cache monitor) is an installable Web application that displays simple cache statistics, cache entries and cache policy information. The CacheMonitor.ear file is available in the <WAS_HOME>/installableApps. For security reasons, you should not use the same host and port as used for your application. Therefore, you first need to create virtual hosts and host aliases for the Cache monitor application:

1. Create a new virtual host called “cache_monitor”. Click **Environment -> Virtual Hosts -> New** in the Administrative Console. Enter **cache_monitor** into the Name field and click **Apply**.
2. Create a new host alias (verify which port numbers are available on your system first) for the cache_monitor host:
 - a. Click **cache_monitor**.
 - b. Click **Host Aliases** in the Additional Properties pane. Click **New**.
 - c. Enter a * (asterisk) into the Host Name field and add **<port_number>** (for example 9070).
3. Click **OK** and save your configuration.
4. Add a new WebContainer Inbound Chain to your application server(s):
 - a. Click **Servers -> Application servers -> <AppServer_Name> -> Web Container Settings -> Web container transport chains**. Click **New**.
 - b. Enter **CacheMonitor** for the Transport chain name and click **Next**.
 - c. Enter * for the Host and **<port_number>** (for example, 9070) for the Port. We also used the port number in the Port name field.
 - d. Click **Next**, check the summary of the settings, click **Finish** and save your configuration.
5. This needs to be done for each application server in the environment. In our scenario we are using three clustered application servers: Web2a and Web2b reside on the same machine, so we need different ports for these two application servers. Web1 resides on a different machine, so we can use the same port number as was used for Web2a. Our configuration is as follows:
 - Port 9070 for Web1 and Web2a
 - Port 9071 for Web2b

Repeat step 4 for each application server in your cluster.

If you have a similar scenario, you also need to add all ports configured in this step to the cache_monitor virtual host that was created in step 2.

We are also using a separate cluster for our EJB containers. We do not need this configuration in the application servers that belong to the EJBcluster.

Now that these preliminary tasks are done, you can install the Cache monitor application. Here is how to do this:

1. Open the Administrative Console and verify whether the Cache monitor is already installed. To do so, click **Applications -> Enterprise Applications**. See Figure 10-5. If CacheMonitor is not displayed in this list, click **Install**.

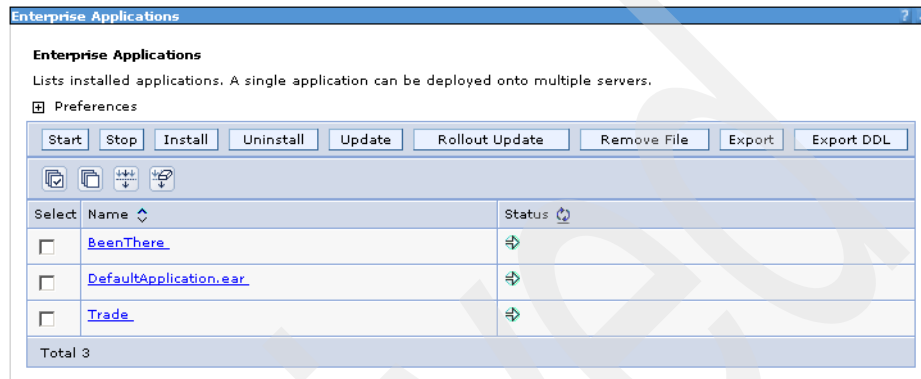


Figure 10-5 Installed Enterprise Applications

2. On the Preparing for the application installation panel (Figure 10-6), browse to the <WAS_HOME>/installableApps Server path and select the **CacheMonitor.ear** file. Click **OK**, then click **Next**.

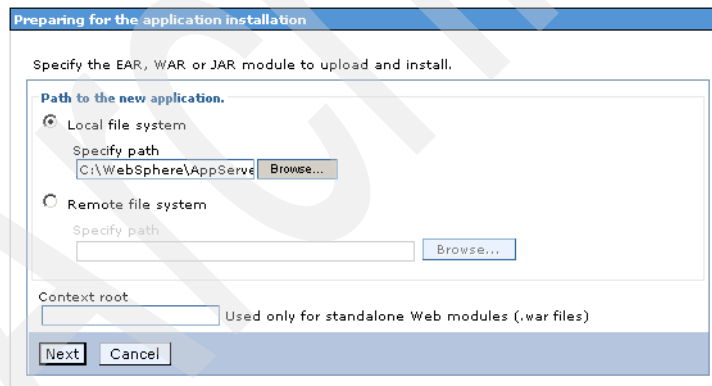


Figure 10-6 Cache monitor installation - Select CacheMonitor.ear

3. On the next panel, enter **cache_monitor** for the Virtual Host (see Figure 10-7 on page 518). Click **Next**.

Preparing for the application installation

Choose to generate default bindings and mappings.

☐ Generate Default Bindings

Override:

☒ Do not override existing bindings

☐ Override existing bindings

Virtual Host

☐ Do not use default virtual host name for Web modules

☒ Use default virtual host name for Web modules:

Host name
cache_monitor

Specific bindings file
 Browse...

Previous Next Cancel

Figure 10-7 Select Virtual Host - cache_monitor

4. On the Application Security Warnings pane, click **Continue**. Accept the default options on the next panel (Step 1) and click **Next** once again. See Figure 10-8 on page 519.

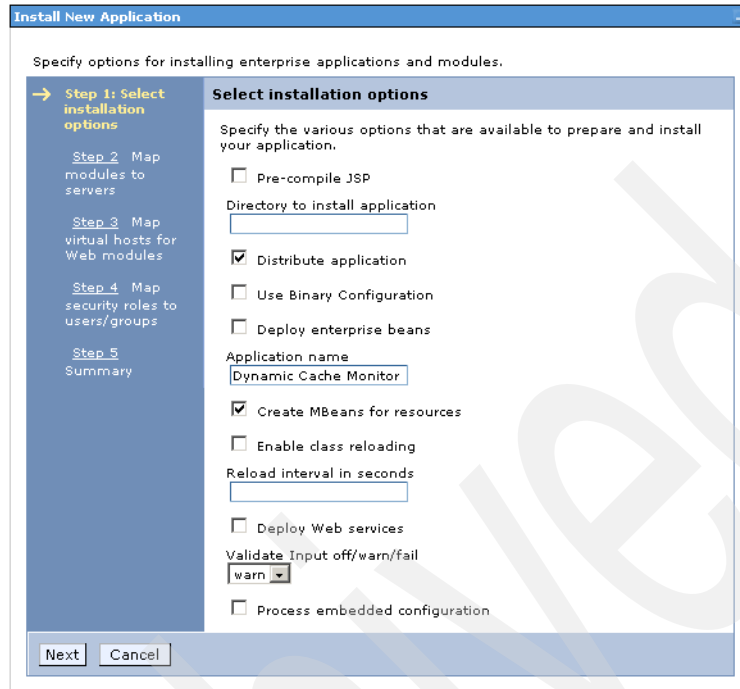


Figure 10-8 Cache monitor installation - Installation options

5. Step 2 - Map modules to servers (Figure 10-9 on page 520).

On this panel, you need to select your application server and the **Dynamic Cache Monitor Module**. Click **Apply**, then click **Next**. You do not need to include the Web servers, select only the Web container cluster.

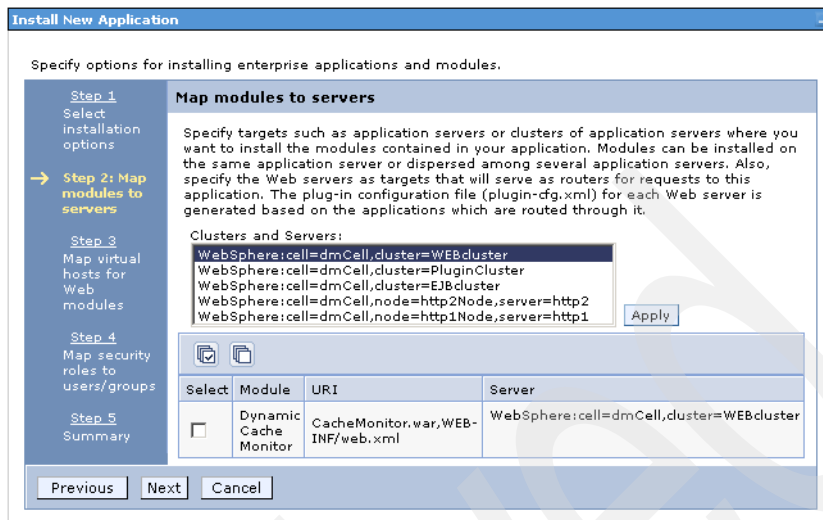


Figure 10-9 Cache monitor installation - Map modules to application servers

6. Step 3- Map virtual hosts for Web modules (Figure 10-10).

We have created a virtual host called “cache_monitor” in step 1 on page 516 which we are now referring to.

Check **Dynamic Cache Monitor** and select the **cache_monitor** Virtual host from the pull-down menu. Then click **Next**.

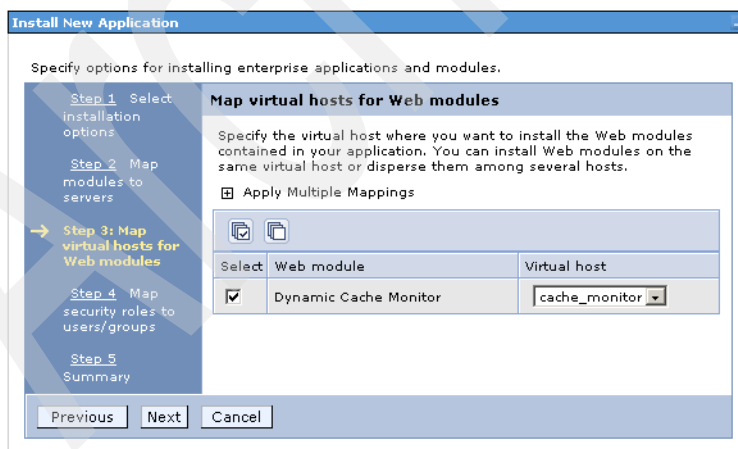


Figure 10-10 Cache monitor installation - Map virtual hosts for Web modules

7. Step 4 - Map security roles to users/group (Figure 10-11 on page 521).

Select **All Authenticated**, then click **Next** once more. Please note that this installation panel is only important if you have security enabled for your application servers.

Install New Application

Specify options for installing enterprise applications and modules.

Step 1 Select installation options

Step 2 Map modules to servers

Step 3 Map virtual hosts for Web modules

→ Step 4: Map security roles to users/groups

Step 5 Summary

Map security roles to users/groups

Each role that is defined in the application or module must map to a user or group from the domain user registry.

Look up users Look up groups

Select	Role	Everyone?	All authenticated?	Mapped users	Mapped groups
<input type="checkbox"/>	administrator	<input type="checkbox"/>	<input checked="" type="checkbox"/>		

Previous Next Cancel

Figure 10-11 Cache monitor installation - Map security roles to users/groups

8. Confirm the installation on the Summary window as shown in Figure 10-12 on page 522, click **Finish** and **Save** your changes to the master configuration.

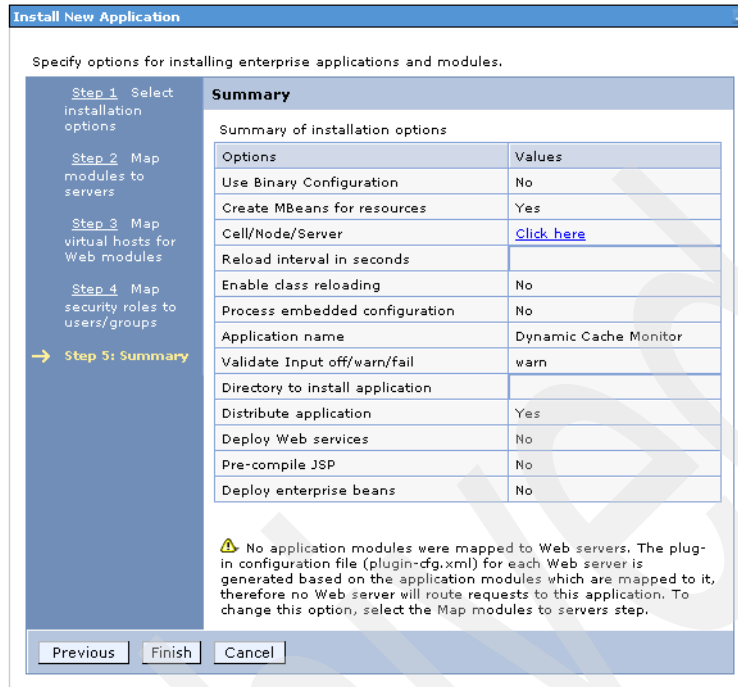


Figure 10-12 Cache monitor installation - Summary

Note: The Summary pane in Figure 10-12 shows an warning related to the fact that we did not associate the application with any Web server. In the case of Cache Monitor, this is the proper way to do it, as we will not use the Web server(s) to access it.

9. Restart your application server(s) - the WEBcluster in our scenario.

You can access the Cache monitor using a Web browser and the URL

`http://<your_appserver_hostname>:<cache_monitor_port_number>/cachemonitor`

We are using these URLs:

- ▶ `http://app1.itso.ibm.com:9070/cachemonitor/` for Web1
- ▶ `http://app2.itso.ibm.com:9070/cachemonitor/` for Web2a
- ▶ `http://app2.itso.ibm.com:9071/cachemonitor/` for Web2b

If your application server is configured for global security, you need to provide a valid user ID and password after invoking the above URL.

Figure 10-13 shows the Cache Monitor start page which allows access to the cache statistics and some configurable parameters. These are configured in the Administrative Console (see “Enabling dynamic cache service” on page 524).

WebSphere Application Server V6 allows you to create several cache instances, so the new version of the Cache Monitor application includes a pull-down box that allows you to select a specific instance.

Statistic	Value
Cache Size	2000
Used Entries	0
Cache Hits	0
Cache Misses	0
LRU Evictions	0
Explicit Removals	0
Default Priority	1
Servlet Caching Enabled	No
Disk Offload Enabled	No

Figure 10-13 WebSphere Cache monitor start page

We explain some functionality of the Cache monitor as we work on our sample scenarios, but please use the WebSphere InfoCenter to fully understand all functionality. The WebSphere Information Center is available at:

<http://www.ibm.com/software/webservers/appserv/infocenter.html>

The ESI processor's cache is also monitored through the Dynamic Cache Monitor application. In order for the ESI processor's cache to be visible in the Cache monitor, the DynaCacheEsi application must be installed and the `esiInvalidationMonitor` property must be set to true in the `plugin-cfg.xml` file. See “External caching by Web server plug-in” on page 566 for more information about the ESI processor.

10.4.2 Enabling dynamic cache service

The dynamic cache service is an in-memory cache system that has disk offload capability. Its caching behavior is defined in an XML file named `cachespec.xml`. A graphical user interface (GUI) tool for building the cache policy file is available, we describe it in detail in 10.3.2, “Dynamic Cache Policy Editor” on page 515.

Unique ID strings distinguish unique entries in WebSphere Application Server’s dynamic content caching solution. These ID strings can be defined declaratively with the XML configuration file or programmatically by the user’s Java program. Please refer to the WebSphere InfoCenter for more information about this.

WebSphere Application Server dynamic cache service can control external caches. Different groups of external caches can be defined, each with its own set of member caches. The interface between WebSphere Application Server and the external cache is the *External Cache Adapter* provided by WebSphere Application Server. We elaborate more on this topic in “WebSphere external caching scenarios” on page 558.

The dynamic cache service includes an alternative feature named disk offload, which stores the overflow cache entries on disk for potential future access. This feature removes the dynamic cache memory constraints experienced in older WebSphere Application Server versions (this was already available in WebSphere V5).

To enable dynamic caching for an application server, follow these steps:

1. Open WebSphere Administrative Console.
2. Click **Servers** -> **Application servers** in the navigation tree.
3. Select your application server.
4. In the Container Settings pane, select **Container Services** -> **Dynamic Cache Service**.
5. Select **Enable service at server startup** in the Startup state field. This setting should be enabled by default.
6. Click **Apply** or **OK**.

All changes made to the dynamic cache service properties take effect after restarting the server.

Figure 10-14 on page 525 shows the configuration panel of the dynamic cache service in the Administrative Console.

This panel allows you to also configure the *disk offload* (see “Configure disk offload” on page 526) and *cache replication* options. We discuss cache replication in 10.5.4, “Cache replication” on page 547.

Configuration

General Properties

☒ Enable service at server startup

* Cache size
2000 entries

* Default priority
1

Recovery settings

☐ Enable disk offload
Offload location:

☐ Flush to disk

Consistency settings

☒ Enable cache replication
Full group replication domain:

Replication type:
Not Shared
Push frequency:
0
[No full group replication domains are defined.](#)

Additional Properties

■ [External cache groups](#)

Apply OK Reset Cancel

Figure 10-14 Enabling dynamic caching in Administrative Console

Configure cache size

As mentioned before, WebSphere Application Server uses JVM memory to store cached objects. Therefore, it is important to know how much memory can be allocated for the cache and based on this information you can set the cache size to the proper value. You need to be able to estimate the total size of objects which can be cached during peak hours. Also, your application designer needs to understand the performance implication of heavy pages.

Tip: If you need to determine the amount of memory needed by the application for caching, you can use a profiling tool such as the one provided with Rational Application Developer. Or you could use Page Detailer to analyze the size of generated pages. See Chapter 15, “Development-side performance and analysis tools” on page 839 for details on these tools.

Configure disk offload

If the estimated total size of all cached objects is bigger than the available memory, you can enable the disk offload option. You then need to configure priorities for the cached objects in the cachespec.xml file. Priority weighting is used in conjunction with the least recently used (LRU) algorithm. Based on this algorithm it is decided which entries are moved from memory to disk if the cache runs out of storage space.

If you decide to use the disk offload option, you need to also configure your file system for fast I/O access. Depending on your hardware and software you can use various disk striping or caching techniques.

Disk offload is configured on the same panel where you enable the dynamic cache service. Refer to 10.4.2, “Enabling dynamic cache service” on page 524 for how to access this panel and Figure 10-14 on page 525 to see how it looks like. These three settings are related to the disk offload configuration:

- ▶ **Enable disk offload**
Specifies whether disk offload is enabled. If a cache entry that was moved to disk is needed again, it is moved back to memory from the file system.
- ▶ **Offload location**
Specifies the location on the disk to save cache entries when disk offload is enabled.

If disk offload location is not specified, the default location, `$install_root/temp/node/servername/_dynacache/cacheJNDIname` is used. If a disk offload location is specified, the node, server name, and cache instance name are appended. For example, `$install_root/diskoffload` generates the location as `$install_root/diskoffload/node/servername/cacheJNDIname`.

This value is ignored if disk offload is not enabled.
- ▶ **Flush to disk**
Specifies if in-memory cached objects are saved to disk when the server is stopped. This value is ignored if Enable disk offload is not selected.

Tuning the disk cache

There are several custom properties for the JVM available to tune the disk cache. All custom properties can be set as follows:

1. In the Administrative Console by selecting **Servers -> Application servers -> <AppServer_Name> -> Java and Process Management -> Process Definition -> Java Virtual Machine -> Custom Properties -> New**.
2. Enter the Name of the custom property.

3. Enter a valid value for the custom property.
4. Save your changes and restart the application server.

There are three custom properties available. The first one is related to the disk cache cleanup time, the other two properties are related to tuning the delay offload function:

- ▶ Tune the disk cache cleanup time using the `com.ibm.ws.cache.CacheConfig.htodCleanupFrequency` custom property.

This property defines the amount of time between disk cache cleanups. By default, the disk cache cleanup is scheduled to run at midnight to remove expired cache entries and cache entries that have not been accessed in the past 24 hours. However, if you have thousands of cache entries that might expire within one or two hours, the files that are in the disk cache can grow large and become unmanageable.

Use `com.ibm.ws.cache.CacheConfig.htodCleanupFrequency` to change the time interval between disk cache cleanups. The value is set in minutes, that is, a value of 60 means 60 minutes between each disk cache cleanup. The default is 0 which means that the disk cache cleanup occurs at midnight every 24 hours.

- ▶ Tune the delay offload function using the `com.ibm.ws.cache.CacheConfig.htodDelayOffloadEntriesLimit` and `com.ibm.ws.cache.CacheConfig.htodDelayOffload` custom properties.

The delay offload function uses extra memory buffers for dependency IDs and templates to delay the disk offload and minimize the input and output operations. However, if most of your cache IDs are longer than 100 bytes, the delay offload function might use too much memory. Use any combination of the following properties to tune your configuration:

- To increase or decrease the in-memory limit, use `com.ibm.ws.cache.CacheConfig.htodDelayOffloadEntriesLimit`.

This custom property specifies the number of different cache IDs that can be saved in memory for the dependency ID and template buffers. Consider increasing this value if you have a lot of memory in your server and you want to increase the performance of your disk cache. The default value is 1000 which means that each dependency ID or template ID can have up to 1000 different cache IDs in memory. Specify a value suitable for your environment, but it must be a number higher than 100 as this is the minimum setting allowed.

- To disable the disk cache delay offload function, use `com.ibm.ws.cache.CacheConfig.htodDelayOffload`.

This custom property specifies if extra memory buffers are used in memory for dependency IDs and templates to delay disk offload and to minimize input and output operations to the disk.

The default value is true which means enabled.

Consider disabling it if your cache IDs are larger than 100 bytes because this option might use too much memory when it buffers your data. If you set this property to false, all the cache entries are copied to disk immediately after they are removed from the memory cache.

10.5 WebSphere dynamic caching scenarios

These are the objectives for our dynamic caching scenarios:

- ▶ Describe and test servlet/JSP caching (see 10.5.1, “Servlet/JSP result caching” on page 529).
- ▶ Describe cache policies for Struts and Tiles caching (see 10.5.2, “Struts and Tiles caching” on page 537).
- ▶ Configure and test command caching (see 10.5.3, “Command caching” on page 540).
- ▶ Configure and test cache replication (see 10.5.4, “Cache replication” on page 547).
- ▶ Explore cache object invalidation (see 10.5.5, “Cache invalidation” on page 556).
- ▶ Explain troubleshooting steps (see 10.5.6, “Troubleshooting the dynamic cache service” on page 557).

Caching of Web services and Web services client caching is covered in Chapter 24, “Web services caching” of the redbook *WebSphere Version 6 Web Services Handbook Development and Deployment*, SG24-6461.

These scenarios explore caching in the application tier as shown in Figure 10-15 on page 529. Refer to 10.6, “WebSphere external caching scenarios” on page 558 for scenarios that explore caching in components out of the application tier.

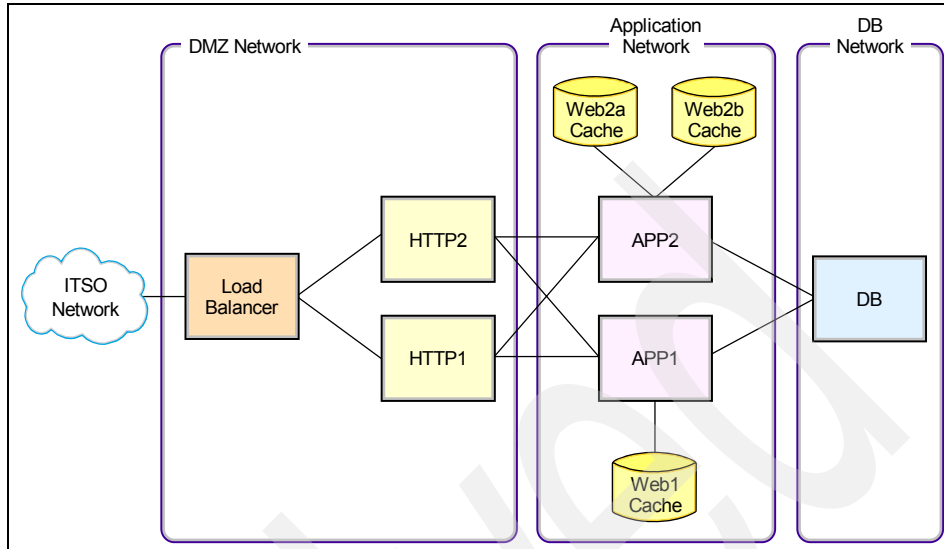


Figure 10-15 Redbook sample dynamic caching infrastructure

10.5.1 Servlet/JSP result caching

Servlet/JSP result caching intercepts calls to a servlet's service method, and checks whether the invocation can be served from cache. If the request cannot be served from cache, the servlet is invoked to generate the output that will be cached. The resulting cache entry contains the output and/or the side effects of the invocation, like calls to other servlets or JSP files. Figure 10-16 on page 530 shows all steps if the output is not cached. If the output is cached, then steps 6, 7 and 8 are skipped.

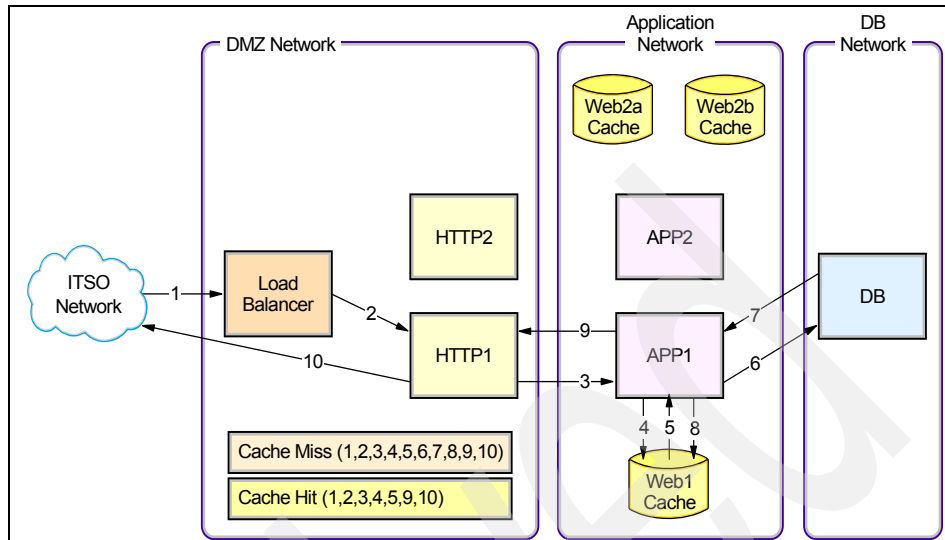


Figure 10-16 Servlet/JSP result caching - infrastructure view

Figure 10-17 gives a closer granularity look on steps 4 and 5. You can see that consequent requests are served only by the Web container, without involving the EJB container (steps 5 to 9).

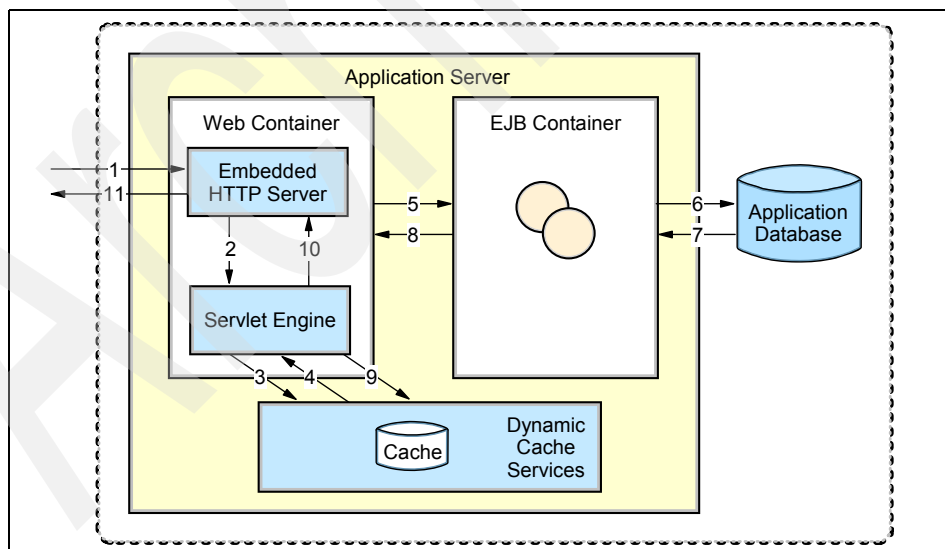


Figure 10-17 Servlet/JSP result caching - application server view

Servlet/JSP result caching can be based on:

- ▶ Request parameters and attributes
- ▶ The URI used to invoke the servlet or JSP
- ▶ Session information
- ▶ Cookies
- ▶ Pathinfo and servlet path
- ▶ Http header and Request method

Servlet/JSP result caching can be used to cache both whole pages or fragments. This concept is explained in Figure 10-18.

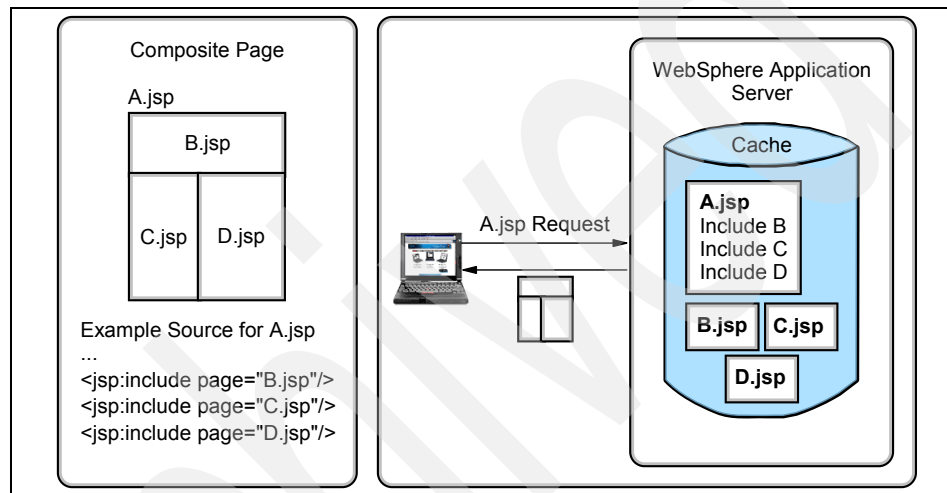


Figure 10-18 Servlet/JSP result caching - fragmentation

Configuration steps

1. Enabling servlet and JSP result caching.

To enable servlet caching, you must enable the dynamic cache service for the Web container by performing these steps:

- a. Open the Administrative Console.
- b. Click **Servers** - > **Application servers** in the console navigation tree.
- c. Select your application server.
- d. Select **Web Container Settings** -> **Web container**.
- e. Select the **Enable servlet caching** check box as shown in Figure 10-14 on page 525.
- f. Click **Apply** or **OK**.

This needs to be done for all application servers. All changes made to the Web container properties take effect after restarting the server.

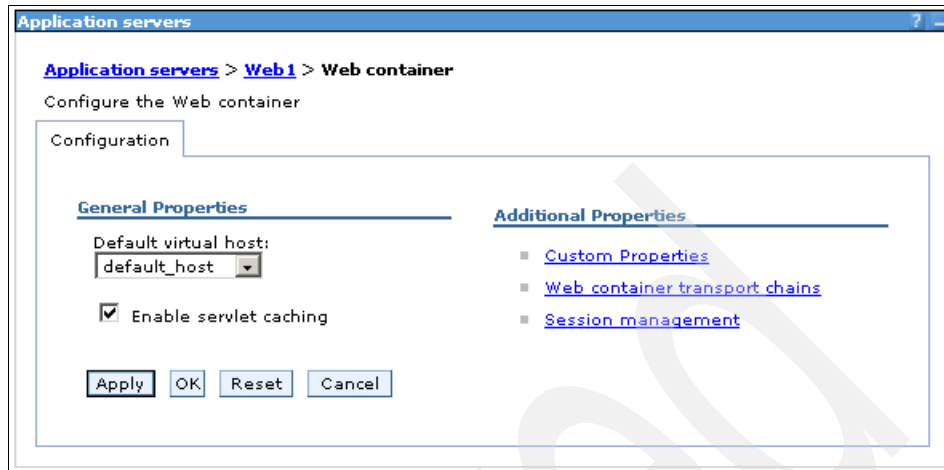


Figure 10-19 Enabling servlet caching

2. Configure cachespec.xml.

In the runtime environment, the cache parses the cachespec.xml file on startup and dynamically whenever the file is modified. The cache extracts from each `<cache-entry>` element a set of configuration parameters. Every time a new servlet or other cacheable object initializes, the cache attempts to match each of the different `<cache-entry>` elements, to find the configuration information for that object. Different cacheable objects have different `<class>` elements. You can define the specific object a cache policy refers to using the `<name>` element.

As mentioned before, we are using Trade 6 for this scenario. The cachespec.xml file for Trade 6 can be found inside the WEB-INF directory of the Web module found in

```
<WAS_HOME>/profiles/<DM_profile>/config/cells/<cell_name>/applications/  
Trade.ear/deployments/Trade/tradeWeb.war/WEB-INF
```

Do not change this file manually in this directory because your changes may be overwritten by a synchronization from the Deployment Manager. Refer to “Distributing the cachespec.xml file” on page 536 for instructions on how to update a changed cachespec.xml file in the Deployment Manager’s repository and synchronize it to the application servers.

Refer to the WebSphere InfoCenter for a detailed description of cachespec.xml file elements. The InfoCenter can be found at

<http://www.ibm.com/software/webservers/appserv/infocenter.html>

Creating and editing Trade 6 cachespec.xml file

This section explains the cache entries for the Trade 6 home servlet (see Figure 10-4 on page 509). This page contains the following cacheable fragments (formatted or non-formatted):

- ▶ Home servlet - Uses `tradehome.jsp` to format an output
- ▶ `marketSummary.jsp` - `tradehome.jsp` includes `marketSummary.jsp`
- ▶ Market summary command
- ▶ Account command
- ▶ Closed order command
- ▶ Holdings command

For the purpose of demonstrating servlet and JSP result caching we create a new `cachespec.xml` file which contains only the home servlet and market summary JSP cache entries. This new `cachespec.xml` will later be extended to include command caching. If you are not interested in the details of how the entries need to be added to `cachespec.xml`, you can use the `cachespec.xml` file that comes with Trade 6 and go directly to the testing sections ("Testing home servlet and market summary JSP" on page 536 and "Testing command caching" on page 546 respectively).

1. Create first the basic structure of the XML file, which is shown in Example 10-1.

Example 10-1 cachespec.xml file first tags

```
<?xml version="1.0"?>
<!DOCTYPE cache SYSTEM "cachespec.dtd">
<cache>
</cache>
```

2. The elements we add to the basic structure need to be located between the `<cache>` and the `</cache>` tags.

The first element we add is the cache-entry element for the home servlet. In this case, the servlet's URI is `/app` so this will be our cache-entry's name element. Also, since this cache-entry is a servlet, the cache-entry class is `"servlet."` See Example 10-2 on page 534.

Example 10-2 Cache entry for home servlet

```
<?xml version="1.0"?>
<!DOCTYPE cache SYSTEM "cachespec.dtd">
<cache>
  <cache-entry>
    <class>servlet</class>
    <name>/app</name>
  </cache-entry>
</cache>
```

Note: In the next steps we do not show the whole cachespec.xml file, only the entries to be added or changed.

- Example 10-3 shows a definition of cache ID generation rules. The home servlet can be cached only when `action=home` and the servlet engine can retrieve the "JSESSIONID" cookie value (a user needs to be logged in). Therefore, one component of the cache ID will be the parameter "action" when the value equals "home". The second parameter will be JSESSIONID which equals to the valid user session ID. The URI for this servlet is `/app?action=home`.

Example 10-3 shows the complete `<cache-entry>` for the home servlet, and the entries in bold are the ones that represent the new cache ID generation rules that need to be added to the home servlet cache entry.

Example 10-3 Cache ID for home servlet

```
<cache-entry>
  <class>servlet</class>
  <name>/app</name>
  <cache-id>
    <component id="action" type="parameter">
      <value>home</value>
      <required>true</required>
    </component>
    <component id="JSESSIONID" type="cookie">
      <required>true</required>
    </component>
  </cache-id>
</cache-entry>
```

- Two dependency IDs are specified for the home servlet: `Account_UserID` and `Holdings_UserID`. Both IDs are used to identify user accounts by the `UserID`. The home servlet is session dependent and `Account_UserID` is used to invalidate the home servlet when either `LoginCommand` or `LogoutCommand` are invoked. Additionally, `Holdings_UserID` is used by

OrderCompletedCommand to invalidate the home servlet upon the order completion. Example 10-4 shows the new <dependency-id> entries, and they must be placed after the last </cache-id> tag and before the </cache-entry> tag of the home servlet shown in Example 10-3 on page 534. See also “Cache replication testing” on page 554.

Example 10-4 Dependency IDs for home servlet

```

<dependency-id>Account_UserID
  <component id="action" type="parameter" ignore-value="true">
    <value>home</value>
    <required>true</required>
  </component>
  <component id="uidBean" type="session">
    <required>true</required>
  </component>
</dependency-id>
<dependency-id>Holdings_UserID
  <component id="action" type="parameter" ignore-value="true">
    <value>home</value>
    <required>true</required>
  </component>
  <component id="uidBean" type="session">
    <required>true</required>
  </component>
</dependency-id>

```

5. A new cache entry is defined for the marketSummary.jsp, which is included by tradehome.jsp, which formats output from the home servlet command. The marketSummary.jsp can be invalidated based on time value (time-driven) or based on the update of data, which are rendered by marketSummary.jsp (event-driven). The ResetTradeCommand invalidates marketSummary.jsp using MarketSummary dependency ID.

Example 10-5 shows the new entry for the marketSummary.jsp, and it must be placed after the </cache-entry> of the home servlet.

Example 10-5 Cache entry for marketSummary.jsp

```

<cache-entry>
  <class>servlet</class>
  <name>/marketSummary.jsp</name>
  <cache-id>
    <priority>3</priority>
    <timeout>180</timeout>
  </cache-id>
  <dependency-id>MarketSummary
</dependency-id>

```

Distributing the cachespec.xml file

The Trade 6 application is shipped with a cachespec.xml file. The following steps describe how to update this file in the Deployment Manager's repository and synchronize it to the application servers using the WebSphere Administrative Console.

1. Click **Applications -> Enterprise Applications**. Select **Trade** and click the **Update** button.
2. In the Application update options pane, select **Single file**.
3. Enter **tradeWeb.war/WEB-INF/cachespec.xml** in the Relative path to file field, and locate the cachespec.xml you edited using the Local file system or the Remote file system options. See Figure 10-20.

☒ **Single file**

Select this option to update an existing file or to add a new file to the application. If the relative path to the file matches an existing path to a file in the installed application, the uploaded file replaces the existing file. If the relative path to the file does not exist in the installed application, the uploaded file is added to the application.

Relative path to file.
tradeWeb.war/WEB-INF/cachespec.xml

Path to the existing file, or to the desired path for the new file.

Upload the new or replacement files.

☒ **Local file system**

Specify path
C:\temp\cachespec.xml

☐ **Remote file system**

Specify path

Figure 10-20 Updating the cachespec.xml file

4. Click **Next**, click **OK** and save the configuration.
5. Restart the Trade application.

Testing home servlet and market summary JSP

1. Log into the Trade 6 application, for example using
http://app1.itso.ibm.com:9080/trade
2. Open the Cache monitor using
http://app1.itso.ibm.com:9070/cachemonitor

3. After you log into the Trade 6 application, you can see that one entry was added to the cache.
4. Click the Home link in the Trade 6 application and use the Cache monitor to verify that both home servlet and marketSummary.jsp are cached. If caching works properly, you can see that new cache entries were added (Figure 10-21).

Statistic	Value
Cache Size	2000
Used Entries	3
Cache Hits	0
Cache Misses	3
LRU Evictions	0
Explicit Removals	0
Default Priority	1
Servlet Caching Enabled	Yes
Disk Offload Enabled	No

Figure 10-21 Servlet/JSP result caching statistics

Home servlet and marketSummary.jsp are in the cache as shown in Figure 10-22.

Template	Cache ID
/trade/app	/trade/app?action=home:JSESSIONID=0000vIZ67Vlq5qLM0n0Hs93e9Hr:vve65hej:requestType=G
/trade/marketSummary.jsp	/trade/marketSummary.jsp:requestType=GET
/trade/marketSummary.jsp	/trade/marketSummary.jsp:requestType=POST

Figure 10-22 Servlet/JSP result caching statistics - Detailed view

All subsequent requests for these two entries will produce a cache hit as long as the objects are valid.

10.5.2 Struts and Tiles caching

Struts and Tiles caching is an extension of servlet and JSP caching. Enabling servlet caching using the Web container setting in the Administrative Console automatically enables Struts and Tiles cache. In addition to enabling the servlet cache, a cache policy is also required to cache a Struts or Tiles response.

Struts

The Struts framework provides the controller component in the MVC-style application. This controller is a servlet called `org.apache.struts.action.ActionServlet.class`. A servlet mapping of `*.do` is added for this Struts `ActionServlet` servlet in the `web.xml` file of the application so that every request for a Web address that ends with `.do` is processed. The `ActionServlet` servlet uses the information in the `struts-config.xml` file to decide which Struts action class is called to actually run the request for the specified resource.

Only one cache policy is supported per servlet in releases prior to WebSphere Application Server V6.0. But, in the case of Struts, every request URI ending in `.do` maps to the same `ActionServlet.class`. Therefore, to cache Struts responses, the cache policy has to be written for the `ActionServlet` servlet based on its servlet path.

For example, consider two Struts actions: `/HelloParam.do` and `/HelloAttr.do`. To cache their responses based on the `id` request parameter, and the `arg` request attribute respectively, the cache policy for WebSphere V5.x must look as shown in Example 10-6:

Example 10-6 Sample Cache policy for a Struts action in WebSphere V5.x

```
<cache-entry>
  <class>servlet</class>
  <name>org.apache.struts.action.ActionServlet.class</name>
  <cache-id>
    <component id="" type="servletpath">
      <value>/HelloParam.do</value>
    </component>
    <component id="id" type="parameter">
      <required>true</required>
    </component>
  </cache-id>
  <cache-id>
    <component id="" type="servletpath">
      <value>/HelloAttr.do</value>
    </component>
    <component id="arg" type="attribute">
      <required>true</required>
    </component>
  </cache-id>
</cache-entry>
```

However, in WebSphere Application Server V6.0 with the support for mapping multiple cache policies for a single servlet, the previous cache policy can be rewritten as shown in Example 10-7 on page 539:

```
<cache-entry>
  <class>servlet</class>
  <name>/HelloParam.do</name>
  <cache-id>
    <component id="id" type="parameter">
      <required>true</required>
    </component>
  </cache-id>
</cache-entry>
<cache-entry>
  <class>servlet</class>
  <name>/HelloAttr.do</name>
  <cache-id>
    <component id="arg" type="attribute">
      <required>true</required>
    </component>
  </cache-id>
</cache-entry>
```

Tiles

Because the Tiles framework is built on the `jsp:include` tag, everything that applies to JSP caching also applies to Tiles. Similar to the `jsp:include` case, the fragments included using the `tiles:insert` tag are cached correctly only if the `flush` attribute is set to `true`. The only extra feature in tiles caching is caching based on the `tiles` attribute. For example, consider the `layout.jsp` template in Example 10-8:

Example 10-8 Tiles: Define template for `layout.jsp`

```
<html>
<body>
  <%String categoryId =
request.getParameter("categoryId")+"test";%>
  <tiles:insert attribute="header">
    <tiles:put name="categoryId" value="<%= categoryId %%" />
  </tiles:insert>
  <TD width="70%" valign="top"><tiles:insert attribute="body"/></TD>
  <TR>
    <TD colspan="2"><tiles:insert attribute="footer"/></TD>
  </TR>
</body>
</html>
```

The nested `tiles:put` tag is used to specify the attribute of the inserted tile. In this template, a tile attribute of `categoryId` is defined and is passed on to the tile

that is inserted in the placeholder for the header. If this `layout.jsp` file is inserted into a JSP file, as shown in Example 10-9, the tile attribute of `categoryId` is passed on to the `header.jsp` file. The `header.jsp` file can use the `<tiles:useAttribute>` tag to retrieve the value of `categoryId`.

Example 10-9 Tiles: Insert header.jsp into layout.jsp

```
<html>
<body>
<tiles:insert page="/layout.jsp?categoryId=1002" flush="true">
  <tiles:put name="header" value="/header.jsp" />
  <tiles:put name="body" value="/body.jsp" />
  <tiles:put name="footer" value="/footer.jsp" />
</tiles:insert>
</body>
</html>
```

To cache the `header.jsp` file based on the value of the `categoryId` attribute, the cache policy shown in Example 10-10 can be used:

Example 10-10 Tiles caching: Cache policy for header.jsp

```
<cache-entry>
  <class>servlet</class>
  <name>/header.jsp</name>
  <cache-id>
    <component id="categoryId" type="tiles_attribute">
      <required>true</required>
    </component>
  </cache-id>
</cache-entry>
```

Other than the difference discussed here, Struts and Tiles caching is very similar to servlet and JSP file caching.

For additional information about how to configure caching for Struts and Tiles, please refer to the InfoCenter at:

http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tdyn_strutstiles.html

10.5.3 Command caching

The Command Cache introduces the next level of granularity to dynamic content caching. Its primary goal is to serve the object's content from cache and thus minimize the execution of remote messages, such as back-end database JDBC calls, or calls to access data at remote non-database servers.

Generally, the Command Cache is used to cache dynamic data that requires back-end data retrieval or additional computation or manipulation. The command cache forms a good synergy with the servlet/JSP result cache, because, in some cases, even caching the most granular, final formatted fragment is not sufficient. For example, Trade 6's home page (Figure 10-2 on page 506) consists of two sets of information: "Account summary" that is highly personalized, and "Market summary" that is generalized information and usable by many users. "Account summary" is highly user sensitive and stored at the back-end server. In this case, it is not effective to cache the final formatted fragment, but rather use command caching.

The Command Cache is caching at the Java application level. To use Command Cache, user applications need to use the caching capabilities of the *WebSphere Command Framework API*. The WebSphere Command Framework is based on the *Command Design Pattern* widely used in Java programming paradigms. Typically, these applications use "setters" to set the command's input states, one or more execute methods, and "getters" to retrieve the results. The results can be either formatted or raw data.

The Command Cache intercepts the execute method call of a command written for Command Cache and checks whether the command object can be served from the cache. If the command object does not exist in the cache, the logic in the execute method is performed and the resulting object is cached.

The caching behavior of Command Cache is defined declaratively with the XML cache policy file, which describes whether and how a command should be cached. Command Cache can be used at the presentation and/or business logic layer in multi-tier environments.

The Command Cache is easy to use. For existing applications that follow a *Model, View, and Controller (MVC)* design pattern, Command Cache can be implemented with minimal impact to existing code.

Enabling command caching

Cacheable commands are stored in the cache for re-use with a similar mechanism as used for servlets and Java Server Pages (JSP) files. However, in this case, the unique cache IDs are generated based on methods and fields present in the command as input parameters.

1. Developers need to implement the following steps during commands development:
 - a. Create a command and define an interface.

The Command interface specifies the most basic aspects of a command.

You must define the interface that extends one or more of the interfaces in the command package. The command package consists of three interfaces:

- TargetableCommand
- CompensableCommand
- CacheableCommand

In practice, most commands implement the TargetableCommand interface, which allows the command to execute remotely. The code structure of a command interface for a targetable command is shown in Example 10-11:

Example 10-11 TargetableCommand interface

```
...
import com.ibm.websphere.command.*;
public interface MyCommand extends TargetableCommand {
    // Declare application methods here
}
```

b. Provide an implementation class for the interface.

Write a class that extends the CacheableCommandImpl class and implements your command interface. This class contains the code for the methods in your interface, the methods inherited from extended interfaces like the CacheableCommand interface, and the required or abstract methods in the CacheableCommandImpl class.

You can also override the default implementations of other methods provided in the CacheableCommandImpl class.

2. cachespec.xml must be configured.

The Trade 6 home page uses the following commands that can be cached:

- Market summary command
- Account command
- Closed order command
- Holdings command

The MarketSummaryCommand is cached and invalidated on a timeout or by ResetTradeCommand, which uses the MarketSummary dependency ID as shown in Example 10-12.

Note: Add the entries listed in the command caching examples below after the </cache-entry> for the marketSummary.jsp shown in Example 10-5 on page 535.

Example 10-12 Cache entry for MarketSummaryCommand

```

<cache-entry>
  <class>command</class>
  <sharing-policy>not-shared</sharing-policy>
  <name>com.ibm.websphere.samples.trade.command.MarketSummaryCommand</name>
  <cache-id>
    <priority>3</priority>
    <timeout>10</timeout>
  </cache-id>
  <dependency-id>MarketSummary
  </dependency-id>
</cache-entry>

```

The AccountCommand is used to cache a user's account information. AccountCommands are invalidated for each individual user when that users' account information, such as their account balance, are updated by a trading operation.

This is shown in Example 10-13:

Example 10-13 Cache entry for AccountCommand

```

<cache-entry>
  <class>command</class>
  <sharing-policy>not-shared</sharing-policy>
  <name>com.ibm.websphere.samples.trade.command.AccountCommand</name>
  <cache-id>
    <component type="method" id="getUserID">
      <required>true</required>
    </component>
    <priority>3</priority>
  </cache-id>
  <dependency-id>Account_UserID
    <component id="getUserID" type="method">
      <required>true</required>
    </component>
  </dependency-id>
  <dependency-id>AllUsers
  </dependency-id>
</cache-entry>

```

The OrderCompletedCommand signifies that a buy or a sell order has completed for an individual user. When an order completes, a user's holdings and account balance have been modified. This invalidates the AccountCommand, HoldingsCommand and OrdersCommand for that user.

Also, it signifies that an order is completed, therefore, on the next request, the ClosedOrderCommand should run to alert the user of any completed orders. The cache-entry for the OrderCompletedCommand performs these invalidations for an individual user on order completion.

Example 10-14 Cache entry for OrderCompletedCommand

```
<cache-entry>
  <class>command</class>
  <sharing-policy>not-shared</sharing-policy>
  <name>com.ibm.websphere.samples.trade.command.OrderCompletedCommand</name>
  <invalidation>Account_UserID
    <component id="getUserID" type="method">
      <required>true</required>
    </component>
  </invalidation>
  <invalidation>Holdings_UserID
    <component id="getUserID" type="method">
      <required>true</required>
    </component>
  </invalidation>
  <invalidation>Orders_UserID
    <component id="getUserID" type="method">
      <required>true</required>
    </component>
  </invalidation>
  <invalidation>ClosedOrders_UserID
    <component id="getUserID" type="method">
      <required>true</required>
    </component>
  </invalidation>
</cache-entry>
```

The `HoldingsCommand` is used to cache a users' stock holdings information. `HoldingsCommands` are invalidated for each individual user when that user's holdings change due to an `OrderCompletedCommand`. It uses the `Holdings_UserID` dependency ID to identify user Accounts by the `userID`. It also uses the `AllUsers` dependency ID to identify all user Accounts for commands which invalidate all Account Holdings cache entries. Refer to Example 10-15.

Example 10-15 Cache entry for HoldingsCommand

```
<cache-entry>
  <class>command</class>
  <sharing-policy>not-shared</sharing-policy>
  <name>com.ibm.websphere.samples.trade.command.HoldingsCommand</name>
  <cache-id>
    <component type="method" id="getUserID">
      <required>true</required>
    </component>
    <priority>3</priority>
  </cache-id>
  <dependency-id>Holdings_UserID
```

```
<component id="getUserID" type="method">
  <required>true</required>
</component>
</dependency-id>
<dependency-id>AllUsers
</dependency-id>
</cache-entry>
```

Update cachespec.xml file

To make sure that the changes you just made to cachespec.xml for the command caching are used by the application, you now need to update and restart Trade 6 once again as described in “Distributing the cachespec.xml file” on page 536.

Enable command caching in Trade 6

Due to a change in the Trade 6 application (compared to the previous Trade3.1 version), we need to enable it to cache commands. This is needed only for command caching and DistributedMap caching. All other component caching is available by default.

1. Access the Trade 6 home page by connecting to the first application server (in our scenario, our first application server is Web1 on server app1.itso.ibm.com, which listens for connections on port 9080):

`http://app1.itso.ibm.com:9080/trade/`

2. On the Trade main page, click the **Configuration** link (above the Go Trade! link).
3. In the Configuration utilities page, click the **Configure Trade run-time parameters** link.
4. In the Trade Configuration page, scroll down to the Caching Type options, select **Command Caching** (see Figure 10-23 on page 546), and click the **Update Config** button at the bottom of the page.

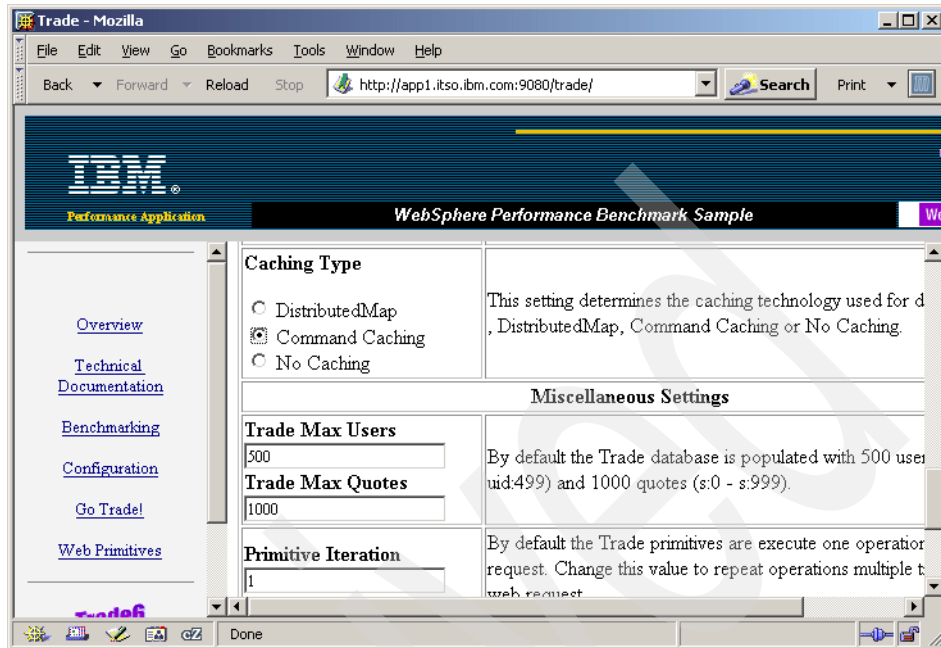


Figure 10-23 Enabling command caching in Trade 6

5. Repeat steps 1 on page 545 through 4 on page 545 for the other application servers in your cluster.

Note: In order to enable command caching for Trade 6 we changed a runtime option. If you restart the Trade application or an application server, you also need to repeat the steps to enable it again.

Testing command caching

We test the four commands included in the home page fragments, plus the home servlet and marketSummary.jsp:

1. Log into the Trade 6 application:
`http://app1.itso.ibm.com:9080/trade/`
2. Open the Cache monitor:
`http://app1.itso.ibm.com:9070/cachemonitor`
3. Clear the cache by clicking the **Clear Cache** button.
4. Click the Home link in the Trade 6 application and use the Cache monitor to verify that several commands, one servlet and one JSP are cached. If the

caching works, you can see up to six cache entries as shown in Figure 10-24 and Figure 10-25.

Statistic	Value
Cache Size	2000
Used Entries	6
Cache Hits	0
Cache Misses	6
LRU Evictions	0
Explicit Removals	0
Default Priority	1
Servlet Caching Enabled	Yes
Disk Offload Enabled	No

Figure 10-24 Command caching - Used Entries

Current Cache Contents
Entries 1 through 6 of 6 < 0 >
<input type="button" value="Clear Cache"/>
Template
/trade/app
/trade/marketSummary.jsp
com.ibm.websphere.samples.trade.command.AccountCommand
com.ibm.websphere.samples.trade.command.ClosedOrdersCommand
com.ibm.websphere.samples.trade.command.HoldingsCommand
com.ibm.websphere.samples.trade.command.MarketSummaryCommand

Figure 10-25 Command caching - Cached entries

10.5.4 Cache replication

The cache replication mechanism allows data to be generated once and then be copied or replicated to other servers in the cluster, thus saving execution time and resources. Caching in a cluster has additional concerns. In particular, the

same data could be required and hence be generated in multiple places. Also, the access to resources needed to generate the cached data can be restricted, preventing access to the data.

Cache replication also aids in cache consistency. If the cache entry is invalidated from one server, the invalidation is propagated to all servers.

We continue to work with the redbook sample infrastructure, now configured for cache replication (Figure 10-26).

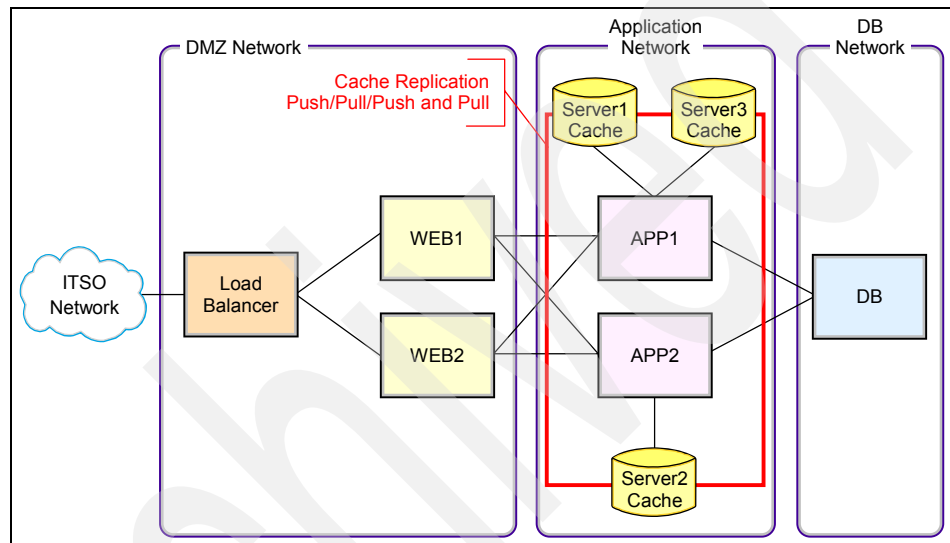


Figure 10-26 Redbook caching replication infrastructure

Enable cache replication

The configuration specific to replication of data can exist as part of the Web container dynamic cache configuration accessible through the Administrative Console, or on a per cache entry basis through the `cachespec.xml` file. This includes the option to configure cache replication at the Web container level, but disabling it for a specific cache entry.

These configuration steps assume that there is already an internal replication domain defined on your system. If this is not true for your environment, then you first need to create these objects.

We recommend creating an exclusive replication domain for the dynamic caching service.

To create a replication domain you need to follow these steps:

1. Click **Environment -> Replication domains -> New** in the WebSphere Administrative Console navigation tree.
2. Enter **DynamicCache** in the Name field and select **Entire Domain** in the Number of replicas pane as shown in Figure 10-27.

Important: Do not use the default value of Single replica for the Number of replicas field when creating a Dynamic Cache replication domain. Only replication domains created with the option Entire Domain will be selectable when configuring cache replication.

3. Click **OK** and save the configuration.

Configuration

General Properties

* Name
DynamicCache

* Request timeout
5

Encryption

Encryption type
none

Regenerate encryption key

Number of replicas

☐ Single replica

☒ Entire Domain

☐ Specify

Number of replicas

Apply OK Reset Cancel

Figure 10-27 Creating a replication domain

Refer to 6.8.7, “Understanding DRS (Data Replication Services)” on page 297 and the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451, for more information about replication domains.

To enable cache replication you need to follow these steps:

1. Click **Servers -> Application servers** in the WebSphere Administrative Console navigation tree.
2. Select your application server.
3. Select **Container Services -> Dynamic Cache Service**.
4. Check the **Enable cache replication** check box.
5. For the Full group replication domain, select **DynamicCache**, which is the replication domain that was created for the dynamic cache service.
6. Select a replication type and enter a value (in seconds) for the Push frequency (a value of 0 means that the content is immediately pushed to other servers in the replication domain). Refer to “Cache replication settings” on page 550 for more information about these options. See Figure 10-28.

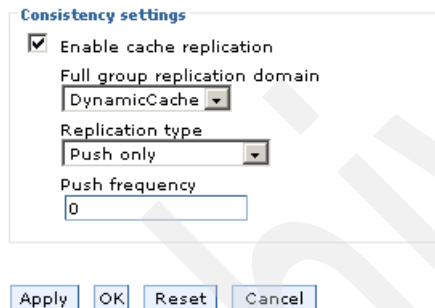


Figure 10-28 Cache replication settings

7. Click **OK**.
8. Repeat steps 1 through 7 for each application server that is part of the DynamicCache replication domain.
9. Save the configuration and restart the application server(s). All changes made to the dynamic cache service properties take effect after restarting the server.

Cache replication settings

The cache replication configuration in WebSphere Application Server V6 allows you to choose a replication type and the push frequency.

Cache replication type

The available cache replication types are:

- Both push and pull

In Both push and pull replication type, cache entries are shared between application servers on demand. When an application server generates a

cache entry, it broadcasts the cache ID of the created entry to all cooperating application servers. Each server then knows whether an entry exists for any given cache ID and for a given request for that entry, the application server knows whether to generate the entry (if no cache ID has been broadcasted for the request) or pull it from the generating application server. These entries cannot store non-serializable data.

In the scenario shown in Figure 10-29, the application server Web2a serves the cacheable/shareable object. It stores it in the Web2a cache. In addition, it also acts as a broker and sends a *cache ID* to the other servers in the cluster (steps A_n). When Web2b receives a request for this published cache ID, it uses the cache ID to retrieve the object from the Web2a cache (steps B_n).

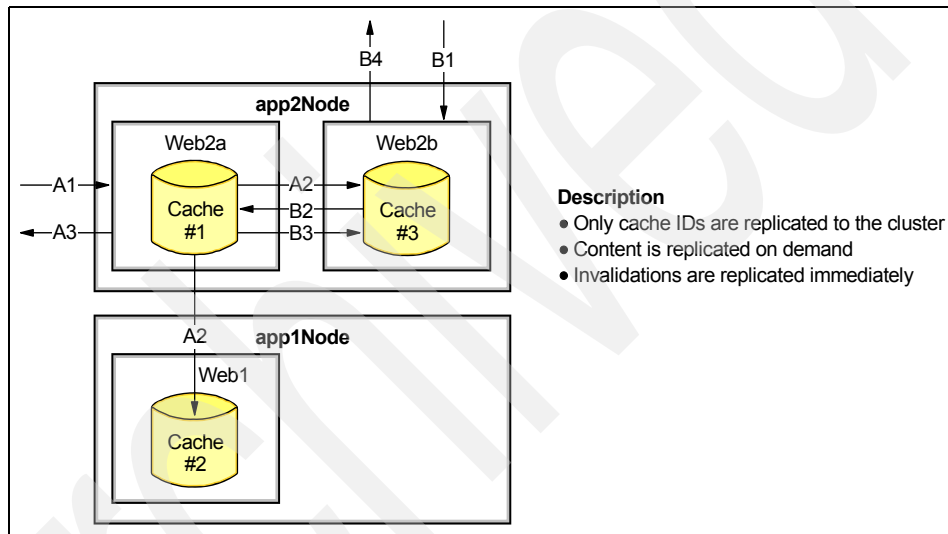


Figure 10-29 Pull and Push cache replication

An advantage of the Both push and pull replication type is that the content is distributed on demand. This has a positive effect on the total number of objects in the cache.

A small disadvantage is that retrieving the content from the remote cache can take longer comparing to the local cache. However, under normal circumstances it is significantly faster than re-generating the content.

► Push only

In Push only replication type, the cache ID and cache content are automatically distributed to the dynamic caches in other application servers or cooperating JVMs. Each cache has a copy of the entry at the time it is created. These entries cannot store non-serializable data.

In the scenario shown in Figure 10-30, Web2a serves the cacheable/shareable object. It stores it in the Web2a cache and it also acts as a broker and sends the *cache ID and the content* to other servers in the cluster (steps A_n). When Web2b receives a request for this cache ID, it retrieves the object from its own cache (steps B_n).

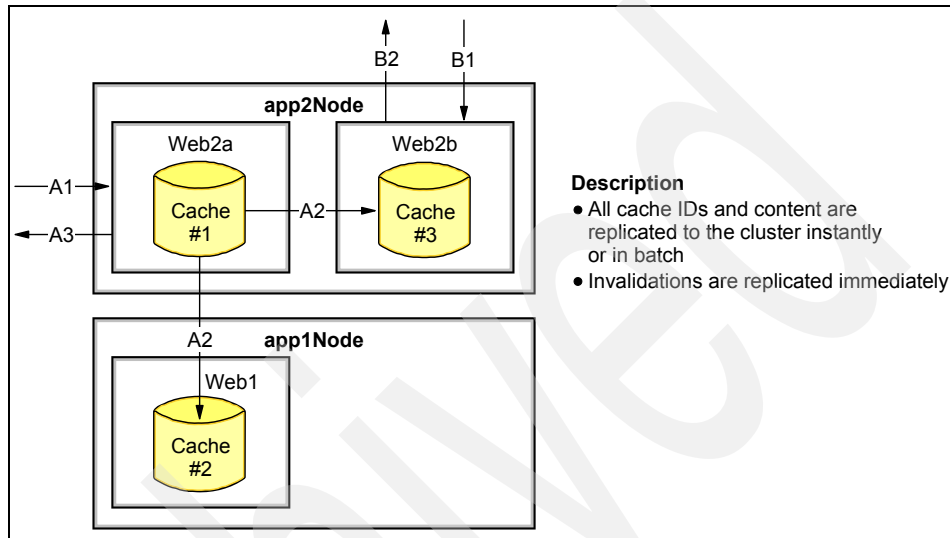


Figure 10-30 Push only cache replication

An advantage of the Push only option is that all caches are synchronized, so even if Web2a crashes, Web2b and Web1 do not need to regenerate the object again.

However, in the case when the total size of cached objects is very high, synchronizing all objects can have a performance impact.

► Not Shared

In Not Shared replication type, cache entries are not shared among different application servers.

Note: To summarize the sharing policy, it is important to understand that the Push only replication type is good for workloads with a high probability of other clusters handling cache hits while Both push and pull replication type is good for unknown or variable probability of cross-cluster cache hits.

Push frequency

Specifies the time in seconds to wait before pushing new or modified cache entries to other servers. A value of 0 (zero) means send the cache entries immediately, and this is the default value.

This option is used by the dynamic cache service to provide a batch update option. Specifically, for the Push only or Both push and pull options, the dynamic cache broadcasts the update asynchronously, based on a timed interval rather than sending them immediately. However, invalidations are always sent immediately. Distribution of invalidations addresses the issue of stale data residing in a cluster.

Setting this property to a value greater than 0 (zero) causes a "batch" push of all cache entries that are created or modified during the time period.

Configure cachespec.xml file for cache replication

The global replication configuration specified in the previous section applies for all caching objects unless the cachespec.xml configuration specifies otherwise. Every entry in cachespec.xml can use the `<sharing-policy>` element, with the values specified below. If the `<sharing-policy>` element is not present, the global settings are used.

The sharing policy on the cache entry level can have the same values as can be defined on the global level:

- ▶ not-shared
- ▶ shared-push
- ▶ shared push-pull

Example 10-16 shows the `marketSummary.jsp` entry in `cachespec.xml`. This JSP will be cached and replicated based on the global settings because the `<sharing-policy>` element has not been specified.

Example 10-16 Replication setting for marketSummary.jsp

```
<cache-entry>
  <class>servlet</class>
  <name>/marketSummary.jsp</name>
  <cache-id>
    <priority>3</priority>
    <timeout>180</timeout>
  </cache-id>
  <dependency-id>MarketSummary
  </dependency-id>
</cache-entry>
```

Example 10-17 contains an entry for MarketSummaryComand. This command will not be replicated, because the sharing-policy element value explicitly specifies that this object cannot be shared among different application servers. This is the same for the other commands too.

Note: Even though this entry is not shared, in a replicated environment invalidations are still sent.

Example 10-17 Replication setting for MarketSummaryCommand

```
<cache-entry>
  <class>command</class>
  <sharing-policy>not-shared</sharing-policy>
  <name>com.ibm.websphere.samples.trade.command.MarketSummaryCommand</name>
  <cache-id>
    <priority>3</priority>
    <timeout>10</timeout>
  </cache-id>
  <dependency-id>MarketSummary
</dependency-id>
</cache-entry>
```

Cache replication testing

We now test the configured replication type Push only. In case you restarted Trade 6 since your last test, you must change the Trade runtime configuration and enable Command Caching on all application servers.

1. Access the Trade 6 application using the first application server:

`http://apl.itso.ibm.com:9080/trade/`

Log in and use Trade for a while (buy, sell and check your portfolio).

2. Access the Cache Monitor running on the same application server:

`http://apl.itso.ibm.com:9070/cachemonitor/`

3. In the Cache Monitor application, click the **Cache Contents** link. Our result is shown in Figure 10-31 on page 555.

Current Cache Contents	
Entries 1 through 6 of 6 < 0 >	
<input type="button" value="Clear Cache"/>	
Template	Cache ID
/trade/app	/trade/app:action=home:JSESSIONID=00034gc
/trade/marketSummary.jsp	/trade/marketSummary.jsp:requestType=GET
com.ibm.websphere.samples.trade.command.AccountCommand	com.ibm.websphere.samples.trade.command.A
com.ibm.websphere.samples.trade.command.ClosedOrdersCommand	com.ibm.websphere.samples.trade.command.C
com.ibm.websphere.samples.trade.command.HoldingsCommand	com.ibm.websphere.samples.trade.command.H
com.ibm.websphere.samples.trade.command.MarketSummaryCommand	com.ibm.websphere.samples.trade.command.M

Figure 10-31 Cache contents of the target application server

You should see a servlet (the home servlet), a JSP (marketSummary.jsp) and several commands (depending on the tasks you performed in Trade). As mentioned in “Configure cachespec.xml file for cache replication” on page 553, all four commands are configured as not-shared, which means that they are not replicated to other application servers in the replication domain.

- In order to check which data was replicated, access the Cache Monitor on the next application server:

<http://app2.itso.ibm.com:9070/cachemonitor/>

- In the Cache Monitor application, click the **Cache Contents** link. This time the entries shown are the ones that were replicated from the first application server, see Figure 10-32. Note that only the shareable elements are in this cache.

Current Cache Contents	
Entries 1 through 2 of 2 < 0 >	
<input type="button" value="Clear Cache"/>	
Template	Cache ID
/trade/app	/trade/app:action=home:JSESSIONID=00034qcm-p4yJIhnHr50lbFfbBH:55142949246946;
/trade/marketSummary.jsp	/trade/marketSummary.jsp:requestType=GET

Figure 10-32 Cache contents of the replicated application server

10.5.5 Cache invalidation

The difference between caching static and dynamic content is the requirement for proactive and effective invalidation mechanisms to ensure the freshness of content. The time-based invalidation alone is no longer adequate for dynamic cache invalidation.

The dynamic cache service provides event-based and time-based invalidation techniques. WebSphere Application Server V6 offers access to programmatic cache and invalidation techniques. Invalidation policies can be defined with XML cache policy files. Invalidation policies allow triggering events to invalidate cache entries without the need to write explicit code. More complex invalidation scenarios may require code, which invokes the invalidation API.

Example 10-18 shows the invalidation policy, which invalidates cache entries for groups of objects using the same Account_UserID dependency ID. For example, home servlet and all commands invoked by home servlet are invalidated when the user logs out from the application.

Example 10-18 Invalidation policy defined in the logout command

```
<cache-entry>
  <class>command</class>
  <sharing-policy>not-shared</sharing-policy>
  <name>com.ibm.websphere.samples.trade.command.LogoutCommand</name>
  <invalidation>Account_UserID
    <component id="getUserID" type="method">
      <required>true</required>
    </component>
  </invalidation>
  <invalidation>Holdings_UserID1
    <component id="getUserID" type="method">
      <required>true</required>
    </component>
  </invalidation>
</cache-entry>
```

An example for time based invalidation can be seen in Example 10-5 on page 535, where marketSummary.jsp has a timeout value of 180 seconds.

Note: In order to use caching safely, you need to make sure that you configure invalidation properly. To do this, you must be familiar with the specifications of the cachespec.xml file and the way how invalidation is handled in your application's code. This chapter provides you with one example, but please refer to the WebSphere InfoCenter for more detailed information about the cachespec.xml file.

If you use cache replication, please note that invalidations are always sent immediately regardless of the share type defined in cachespec.xml or the replication type selected in the Dynamic Cache Services settings.

10.5.6 Troubleshooting the dynamic cache service

In case you are experiencing problems related to the WebSphere Dynamic Cache service, you should follow these steps to resolve your problem:

1. Use the Cache monitor to test.
2. Verify cachespec.xml file.
3. Review the JVM logs for your application server. Messages prefaced with DYNA result from dynamic cache service operations.

- a. View the JVM logs for your application server. Each server has its own JVM log file. For example, if your server is named Web1, the JVM log is located in the subdirectory `<install_root>/logs/Web1/`.

Alternatively, you can use the Administrative Console to review the JVM logs, click **Troubleshooting -> Logs and Trace -> <AppServer_Name> -> JVM Logs -> Runtime tab -> View**.

- b. Find any messages prefaced with DYNA in the JVM logs, and write down the message IDs. A sample message having the message ID DYNA0030E follows:

DYNA0030E: "property" element is missing required attribute "name".

- c. Find the message for each message ID in the WebSphere Application Server InfoCenter. In the InfoCenter navigation tree, click **<product_name> -> Reference -> Troubleshooter -> Messages -> DYNA** to view dynamic cache service messages.

- d. Try the solutions stated under User Response in the DYNA messages.

4. Enable tracing on the application server for `com.ibm.ws.cache.*`. To enable it, click **Troubleshooting -> Logs and Trace -> <AppServer_Name> -> Diagnostic Trace** in the WebSphere Administrative Console. Click the **Configuration** tab, click the **Enable Log** checkbox and click **OK**.

Click **Troubleshooting -> Logs and Trace -> <AppServer_Name> -> Change Log Detail Levels**, click the **Configuration** tab and add the trace string `com.ibm.ws.cache.*=all`. Separate entries with a colon (:). See Figure 10-33 on page 558.

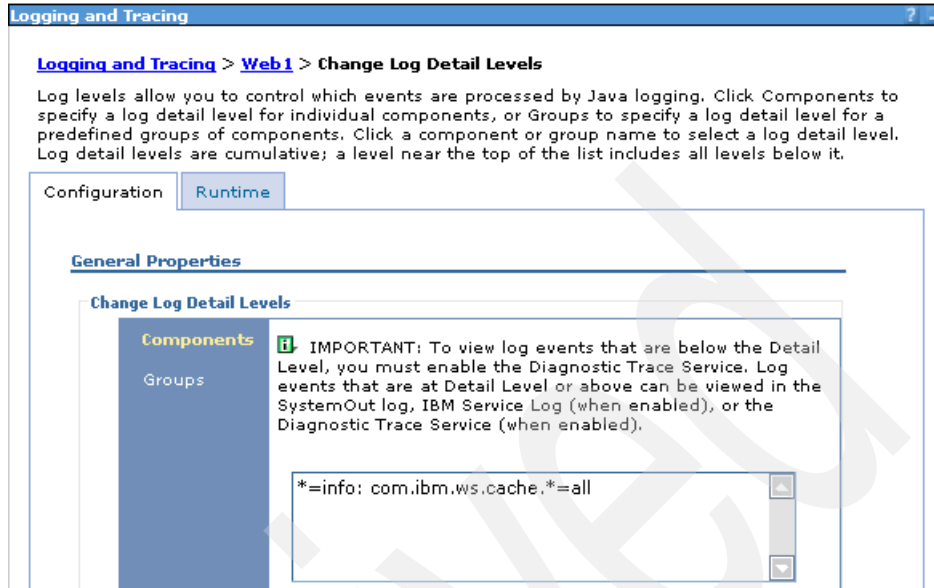


Figure 10-33 Turning on cache trace

Note that `*=info` is the default value, so we recommend keeping it in the configuration.

If you experience problems with cache replication, then you need to enable the trace for `com.ibm.ws.drs.*`.

You need to restart the server and monitor the trace file.

10.6 WebSphere external caching scenarios

The objectives of our external caching scenarios are:

- ▶ Describe the external caching options
- ▶ Configure and test external caching options
- ▶ Explore cache invalidation

We use the following external caches, which can be controlled by the WebSphere dynamic cache service:

- ▶ Web server plug-in

In addition to the well known Web server plug-in functionalities (such as failover and load balancing), the Web server plug-in is integrated with the WebSphere Dynamic Cache to provide in-memory caching of servlets and

JSP pages. It uses the ESI fragment assembly, which further enhances this caching ability with on-the-fly reassembly of JSP pages.

- IBM HTTP Server's high-speed cache

This cache is referred to as the *Fast Response Cache Accelerator* (FRCA) to cache whole pages and fragments.

- Edge Components of IBM WebSphere Application Server Network Deployment V6

The Edge Components can also be configured as WebSphere Application Server's external cache for whole page caching. In this case, the dynamic cache service can be enabled to match pages with their universal resource identifiers (URIs) and export matching pages to the external cache (Figure 10-34). The contents can then be served from the external cache instead of the application server to significantly save resources and improve performance.

Figure 10-34 and Figure 10-35 on page 560 show two examples of exporting a dynamic cache page from the WebSphere Application Server to the external cache.

The first example (Figure 10-34) shows caching of whole pages, where page A.jsp consists of three fragments. These three JSPs are included at compilation time. The event-driven invalidation causes the invalidation of A.jsp from the external cache as a result of any update to B.jsp, C.jsp, or D.jsp.

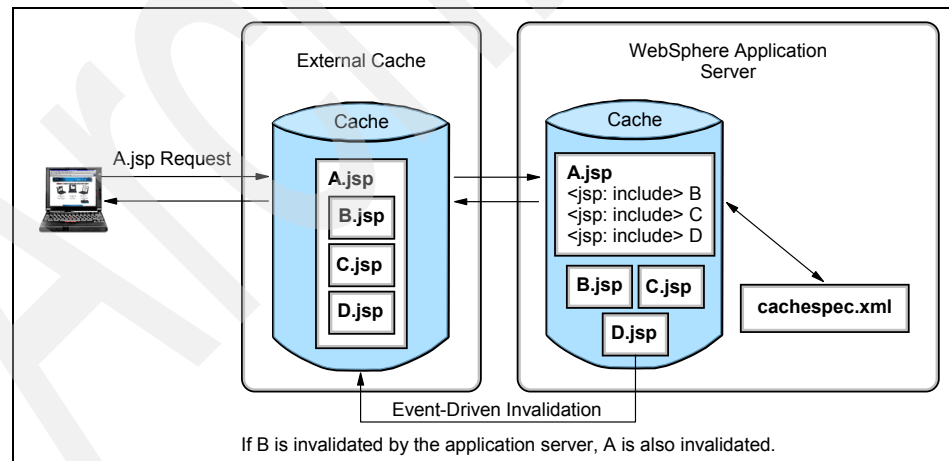


Figure 10-34 Whole page external caching

The second example (Figure 10-35 on page 560) shows caching of a whole page and its' fragments using Edge Side Include (ESI). In this case, if a fragment (for

example B.jsp) is invalidated by event-driven or time-driven invalidation, then the page A.jsp is not invalidated and ESI will tell the cache server to fetch only the invalidated fragment (B.jsp) from the application server. Then, the external cache using the ESI processor assembles a new page A.jsp which consists of the old fragments C.jsp and D.jsp and the new fragment B.jsp.

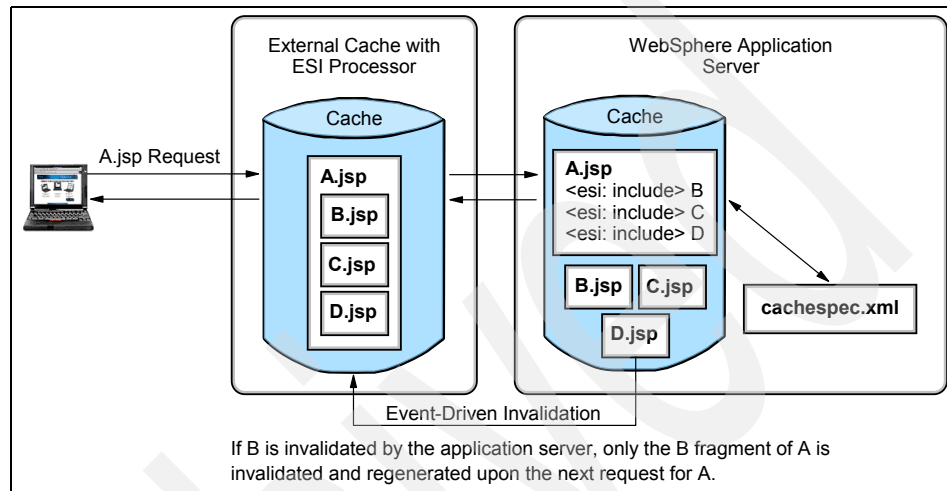


Figure 10-35 Fragment external caching

For example, the Trade 6 home servlet contains the fragment `marketSummary.jsp`. If we invalidate `marketSummary.jsp`, only this JSP will be regenerated upon the next request for the home servlet.

Any servlet and JSP file content that is private, requires authentication, or uses SSL should not be cached externally. The authentication required for those servlet or JSP file fragments cannot be performed on the external cache. A suitable timeout value should be specified if the content is likely to become stale.

10.6.1 WebSphere External Cache configuration

While working on our scenarios we need to perform a few configurations. We explain these configuration steps in this section at the generic level and refer back to this section from each scenario.

Installing the DynaCacheEsi application

First we need to install the DynaCacheEsi application that is needed for ESI caching (and invalidation) in the Web server plug-in. Here is how to do this:

1. Open the Administrative Console and verify whether the DynaCacheEsi application is already installed. To do so, click **Applications** -> **Enterprise Applications**. See Figure 10-5 on page 517. If DynaCacheEsi is not displayed in this list, click **Install**.
2. On the Preparing for the application installation panel (Figure 10-6 on page 517), browse to the <WAS_HOME>/installableApps Server path, select the **DynaCacheEsi.ear** file. Click **OK**, then click **Next**.

Path to the new application.

☐ Local file system

Specify path

☒ Remote file system

Specify path

Context root
 Used only for standalone Web modules (.war files)

Figure 10-36 DynaCacheEsi installation - Select DynaCacheEsi.ear

3. On the next panel, enter **default_host** for the Virtual Host. This application needs to be installed using the same virtual host as the Trade 6 application. Click **Next** -> **Continue**.

☐ Generate Default Bindings

Override:

☒ Do not override existing bindings

☐ Override existing bindings

Virtual Host

☐ Do not use default virtual host name for Web modules

☒ Use default virtual host name for Web modules:

Host name

Specific bindings file

Figure 10-37 Select Virtual Host - default_host

4. On the Application Security Warnings pane, click **Continue**. Accept the default options on the next panel (Step 1) and click **Next** once again. See Figure 10-8 on page 519.

Step 1: Select installation options

Step 2 Map modules to servers

Step 3 Map virtual hosts for Web modules

Step 4 Summary

Select installation options

Specify the various options that are available to prepare and install your application.

☐ Pre-compile JSP

Directory to install application

☒ Distribute application

☐ Use Binary Configuration

☐ Deploy enterprise beans

Application name

☒ Create MBeans for resources

☒ Enable class reloading

Reload interval in seconds

☐ Deploy Web services

Validate Input off/warn/fail

☐ Process embedded configuration

Next Cancel

Figure 10-38 DynaCacheEsi installation - Installation options

5. Step 2 - Map modules to application servers (Figure 10-9 on page 520).

On this panel, you need to select your application server cluster (in our scenario, we selected WEBcluster) and the Web servers, and click the checkbox in the Select column for the **DynaCacheEsi** Module. Click **Apply**, then click **Next**.

Step 1 Select installation options

→ **Step 2: Map modules to servers**

Step 3 Map virtual hosts for Web modules

Step 4 Summary

Map modules to servers

Specify targets such as application servers or clusters of application servers where you want to install the modules contained in your application. Modules can be installed on the same application server or dispersed among several application servers. Also, specify the Web servers as targets that will serve as routers for requests to this application. The plug-in configuration file (plugin-dfg.xml) for each Web server is generated based on the applications which are routed through it.

Clusters and Servers:

```
WebSphere:cell=dmCell,cluster=WEBcluster
WebSphere:cell=dmCell,cluster=PluginCluster
WebSphere:cell=dmCell,cluster=EJBcluster
WebSphere:cell=dmCell,node=http2Node,server=http2
WebSphere:cell=dmCell,node=http1Node,server=http1
```

Apply

Select	Module	URI	Server
<input type="checkbox"/>	DynaCacheEsi	DynaCacheEsi.war,WEB-INF/web.xml	WebSphere:cell=dmCell,cluster=WEBcluster WebSphere:cell=dmCell,node=http2Node,server=http2 WebSphere:cell=dmCell,node=http1Node,server=http1

Previous Next Cancel

Figure 10-39 DynaCacheEsi installation - Map modules to application servers

6. Step 3- Map virtual hosts for Web modules (Figure 10-10 on page 520).

Check **DynaCacheEsi** and select the **default_host** Virtual Host from the pull-down menu. Then click **Next**.

Step 1 Select installation options

Step 2 Map modules to servers

→ **Step 3: Map virtual hosts for Web modules**

Step 4 Summary

Map virtual hosts for Web modules

Specify the virtual host where you want to install the Web modules contained in your application. You can install Web modules on the same virtual host or disperse them among several hosts.

☒ Apply Multiple Mappings

Select	Web module	Virtual host
<input type="checkbox"/>	DynaCacheEsi	default_host

Previous Next Cancel

Figure 10-40 DynaCacheEsi installation - Map virtual hosts for Web modules

7. Confirm the installation on the Summary window as shown in Figure 10-12 on page 522, click **Finish** and **Save** your changes to the master configuration.

Step 1 Select installation options

Step 2 Map modules to servers

Step 3 Map virtual hosts for Web modules

→ Step 4: Summary

Summary

Summary of installation options

Options	Values
Use Binary Configuration	No
Create MBeans for resources	Yes
Cell/Node/Server	Click here
Reload interval in seconds	5
Enable class reloading	Yes
Process embedded configuration	No
Application name	DynaCacheEsi
Validate Input off/warn/fail	warn
Directory to install application	
Distribute application	Yes
Deploy Web services	No
Pre-compile JSP	No
Deploy enterprise beans	No

Previous

Finish

Cancel

Figure 10-41 DynaCacheEsi installation - Summary

External cache group settings

You need to define an external cache group controlled by WebSphere Application Server. To do so:

1. Start the WebSphere Administrative Console and click **Servers -> Application servers**.
2. Select your application server from the list.
3. Select **Container Services -> Dynamic Cache Service**.
4. Select **External Cache Groups** from the Additional Properties pane.

This panel (shown in Figure 10-42 on page 565) allows you to create, delete or update an existing External Cache Group.

5. Click **New**. This launches the Configuration window where you can specify the name of the external cache group.

Note: The external cache group name needs to match the external cache property defined for servlets or JSPs in the cachespec.xml file.

We need to create three different external cache groups for our scenarios:

- AFPA - which is the adapter used for the Fast Response Cache Accelerator

- EsiInvalidator - used for the Web server plug-in. Normally, the EsiInvalidator external cache group exists by default.
- ITSO-APPSERV - used for the Caching Proxy

Tip: In a production environment, you need to create the external cache groups on all application servers that are serving the cached application. To simply test the function and to learn how to set it up in our test scenario, it is sufficient to create and configure them on two application servers, for example, on Web1 and Web2a and to stop additional application servers that are not configured properly.

At this point, you only need to create the three entries, you do not need to configure them yet. We will explain them in more details in the next sections.

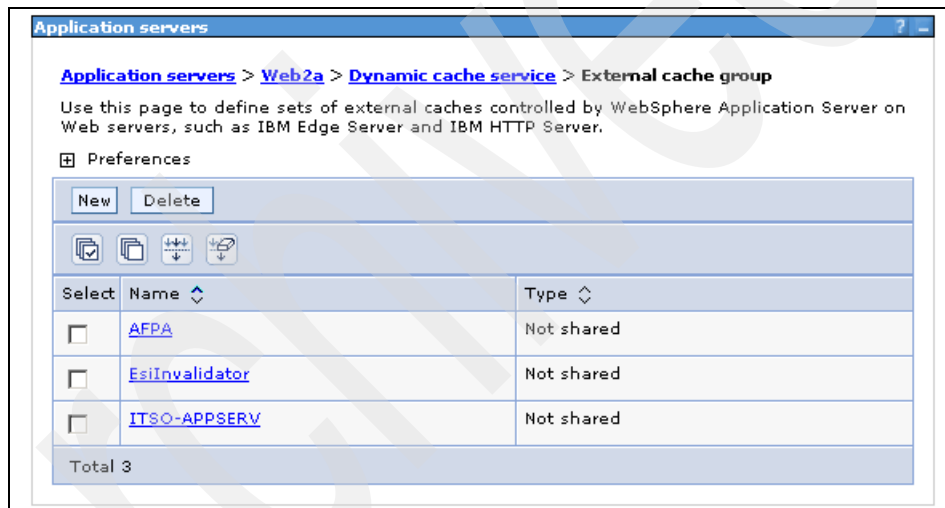


Figure 10-42 External Cache Groups

When external caching is enabled, the cache matches pages with its URIs and pushes matching pages to the external cache. The entries can then be served from the external cache instead of the application server.

External cache group member settings

Once you have created the three external groups, you need to configure the specific caches that are members of each cache group.

1. Open the Administrative Console and click **Servers -> Application servers**.
2. Select your application server.

3. Select **Container Services -> Dynamic Cache Service**, then **External cache groups**.
4. Click **<external_cache_group_name>** followed by **External cache group members** from the Additional Properties.
5. Click **New** to launch the Configuration panel.

There are two values that you need to specify on the Configuration panel:

- Address
Specifies a configuration string used by the external cache adapter bean to connect to the external cache.
- Adapter bean name
Specifies an adapter bean specific for the external cache member.

You can configure three different types of external cache group members.

- ▶ Fast Response Cache Accelerator (called AFPA in our example):
 - Address: Port (port on which AFPA listens)
 - Adapter Bean Name: `com.ibm.ws.cache.servlet.Afpa`
- ▶ ESI (called EsIInvalidator in our scenario):
 - Address: Hostname (and possibly port number)
 - Adapter Bean Name:
`com.ibm.websphere.servlet.cache.ESIInvalidatorServlet`
- ▶ WTE - Caching Proxy (called ITSO-APPSERV in our example):
 - Address: `hostname:port` (hostname and port on which WTE is listening)
 - Adapter Bean Name: `com.ibm.websphere.edge.dynacache.WteAdapter`

10.6.2 External caching by Web server plug-in

WebSphere Application Server leverages the Edge Side Includes (ESI) specification to enable caching and assembly of distributed fragments.

Edge Side Includes is a simple mark-up language used to define Web page fragments for dynamic assembly of a Web page at the edge of network. ESI is an open standard specification supported by Akamai and leaders in the Application Server, Content Management Systems, and Content Delivery Networks markets.

With the Distributed Fragment Caching and Assembly Support, WebSphere Application Server customers can defer page assembly to any ESI compliant surrogate server, such as Akamai EdgeSuite service. This may result in a

significant performance advantage if fragments can be cached and reused at the surrogate.

WebSphere Application Server provides distributed fragment caching and assembly support through the Web server plug-in. WebSphere Application Server uses the Web server plug-in to communicate with the HTTP Server. This plug-in has the ability to cache whole pages or fragments. Additionally, it can dynamically assemble Web pages containing ESI <esi:include> tags. Any server with WebSphere HTTP plug-in support can gain the benefits provided by the WebSphere Application Server dynamic cache service. Figure 10-43 show the infrastructure topology we used for this scenario. As you can see here, the edge cache resides on the Web server machines:

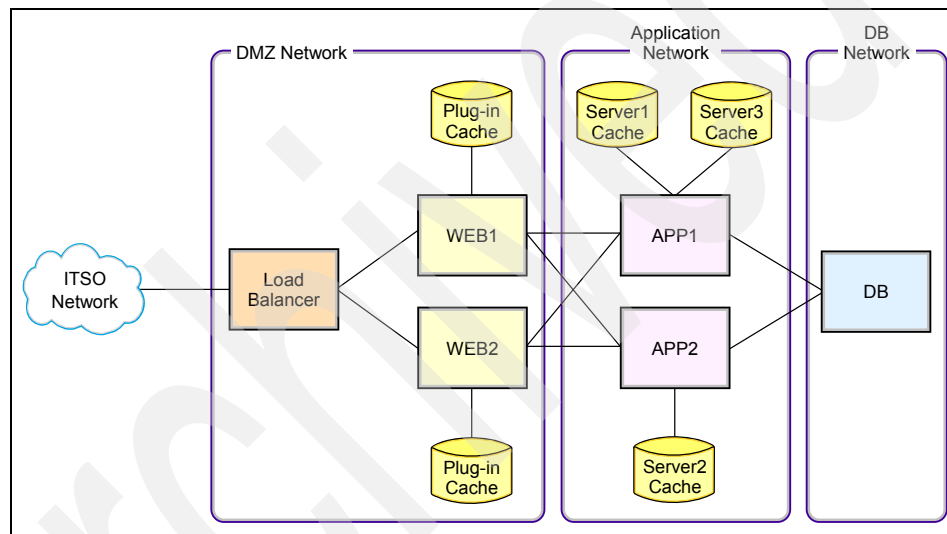


Figure 10-43 ITSO Web server plug-in infrastructure

With dynamic cache service's external cache control, distributed fragment caching and assembly support, dynamic content can be exported, cached, and assembled at the most optimal location, closer to the end user. More important, WebSphere Application Server can maintain control of the external cache through its Invalidation Support to ensure the freshness of cached content. As a result, WebSphere Application Server customers are equipped to create and serve highly dynamic Web pages without jeopardizing page performance and user experiences.

In "Edge Side Includes" on page 614 we provide some existing ESI benchmarks for the Trade 3 application.

ESI Processor

The cache implemented by the ESI processor is an in-memory cache, not a disk cache. Therefore, the cache entries are not saved when the Web server is restarted.

The basic operation of the ESI processor is as follows: When a request is received by the Web server plug-in, it is sent to the ESI processor, unless the ESI processor is disabled. It is enabled by default. If a cache miss occurs, a Surrogate-Capabilities header is added to the request by the plug-in and the request is forwarded to the WebSphere Application Server. If the dynamic servlet cache is enabled in the application server, and the response is edge cacheable, the application server returns a Surrogate-Control header in response to the WebSphere Application Server plug-in.

The value of the Surrogate-Control response header contains the list of rules that are used by the ESI processor in order to generate the cache ID. The response is then stored in the ESI cache, using the cache ID as the key. For each ESI include tag in the body of the response, a new request is processed such that each nested include results in either a cache hit or another request forwarded to the application server. When all nested includes have been processed, the page is assembled and returned to the client.

Configuration steps

1. Configure the external cache group on the application servers (in our case for the WEBcluster servers). Follow the steps outlined in “WebSphere External Cache configuration” on page 560 and create an external cache group named “EsiInvalidator” if it is not there by default. Then add a member with these parameters:
 - Address: localhost
 - Adapter bean name:
com.ibm.websphere.servlet.cache.ESIInvalidatorServlet

The screenshot shows a configuration window for an external cache group. At the top are 'New' and 'Delete' buttons. Below them are icons for adding, removing, and refreshing members. The main area is a table with columns 'Select', 'Address', and 'Adapter bean name'. One member is listed with 'localhost' as the address and 'com.ibm.websphere.servlet.cache.ESIInvalidatorServlet' as the adapter bean name. A 'Total 1' summary is at the bottom.

Select	Address	Adapter bean name
<input type="checkbox"/>	localhost	com.ibm.websphere.servlet.cache.ESIInvalidatorServlet

Total 1

Figure 10-44 EsiInvalidator external group settings

Note: In our tests, the `EsIInvalidator` was already created in all application servers, containing the member `localhost`, as shown in Figure 10-44. In this case, you can keep the original configuration.

2. Configure the ESI processor.

The ESI processor configuration is located in the WebSphere Web server plug-in configuration file (`plugin-cfg.xml`). Example 10-19 shows the ESI configuration options (they are usually located at the beginning of the file).

Example 10-19 ESI configuration options in `plugin-cfg.xml` file

```
<Property Name="ESIEnable" Value="true"/>
<Property Name="ESIMaxCacheSize" Value="1024"/>
<Property Name="ESIInvalidationMonitor" Value="false"/>
```

The `esiMaxCacheSize` specifies the maximum size of the cache in 1 KB units. The default maximum size of the cache is 1 MB. If the cache is full, the first entry to be evicted from the cache is the entry that is closest to expiration.

The `esiInvalidationMonitor` parameter specifies whether or not the ESI processor should receive invalidations from the application server.

3. In order to enable an application server to send an explicit invalidation, the `esiInvalidationMonitor` property from Example 10-19 must be set to true and the `DynaCacheEsi` application must be installed on the application server (refer to “Installing the `DynaCacheEsi` application” on page 560).

If the `esiInvalidationMonitor` property is set to true but the `DynaCacheEsi` application is not installed, then errors will occur in the Web server plug-in and the request will fail.

In order to configure it, open the WebSphere Administrative Console and click **Web servers -> <WebServer_Name> -> Plug-in properties -> Caching**. The **Enable Edge Side Include (ESI) processing to cache the responses** is enabled by default. Select the **Enable invalidation monitor to receive notifications** checkbox and click **OK**, as shown in Figure 10-45 on page 570.

Repeat this step for all Web servers that provide access to the application server cluster.

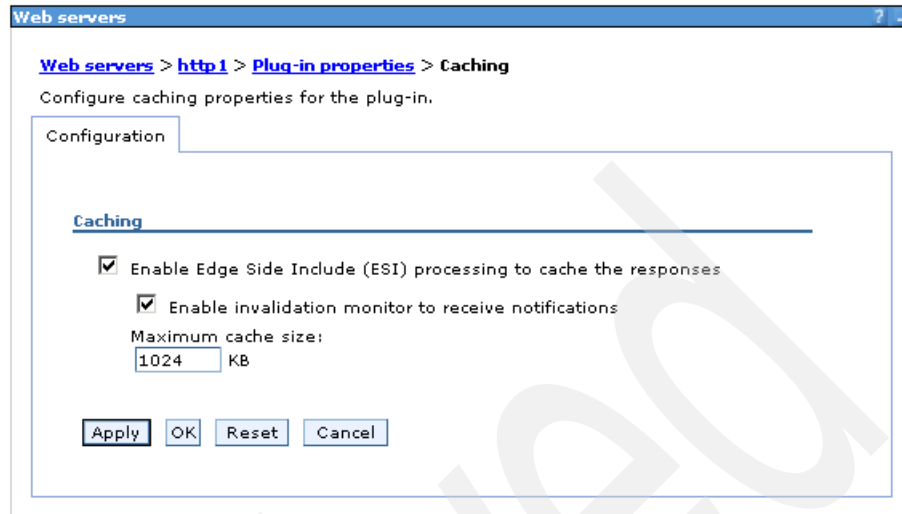


Figure 10-45 Enabling Edge Side Includes

Save the configuration, regenerate and redistribute the plug-in configuration file if it is not configured to be regenerated and distributed automatically.

4. Add a cache policy in the cachespec.xml file for the servlet or JSP file you want to be cached in the edge cache. To do so, Add `<property name="EdgeCacheable">true</property>` for those entries, as shown in Example 10-20 for marketSummary.jsp. This property is set as needed in the default cachespec.xml for Trade 6.

Example 10-20 Edge cacheable property for marketSummary.jsp

```
<cache-entry>
  <class>servlet</class>
  <name>/marketSummary.jsp</name>
  <property name="EdgeCacheable">true</property>
  <cache-id>
    <priority>3</priority>
    <timeout>180</timeout>
  </cache-id>
  <dependency-id>MarketSummary
  </dependency-id>
</cache-entry>
```

5. When WebSphere Application Server is used to serve static data, such as images and HTML on the application server, the URLs are also cached in the ESI processor. This data has a default timeout of 300 seconds. You can change the timeout value by adding the property

`com.ibm.servlet.file.esi.timeOut` to your JVM's command-line parameters.

Testing ESI external caching

Follow these steps to test dynamic caching in the Web server plug-in:

1. Stop and start your Web server(s). We noticed that simply reloading the `plugin-cfg.xml` file is not enough and a real stop and start of the Web server(s) is necessary.
2. Start the `DynaCacheEsi` application if not done so already.
3. Go Trade, for example using:

```
http://cproxy.itso.ibm.com/trade
http://cluster.itso.ibm.com/trade
http://http1.itso.ibm.com/trade
```

In order to test the Web server plug-in dynamic caching configuration, the most important thing is that you access the application through the plug-in (in our case through the caching proxy, or the cluster, or on a Web server) and not directly on the application server.

Execute a few tasks in Trade.

4. Monitor the ESI cache as outlined below.

ESI cache monitoring

The ESI processor's cache is monitored through the Dynamic Cache Monitor application. In order for ESI processor's cache to be visible in the Cache monitor, the `DynaCacheEsi` application must be installed and the `esiInvalidationMonitor` property must be set to true in the `plugin-cfg.xml` file.

Open the Dynamic Cache Monitor application. During our tests we found that the Edge statistics are only displayed in one of the Cache Monitor applications on one of the application servers so you might need to verify which one is the correct one.

To get this statistic you need to select **Edge Statistics** in the Cache monitor navigation bar.

The Edge Statistics could be confused with the Caching Proxy statistics, which is part of IBM Edge Components. However, the term Edge statistics in this case relates to the ESI cache statistics. The ESI cache statistics are shown in Figure 10-46 on page 572.

If you do not see Edge Statistics right away, try clicking the **Refresh Statistics** button.

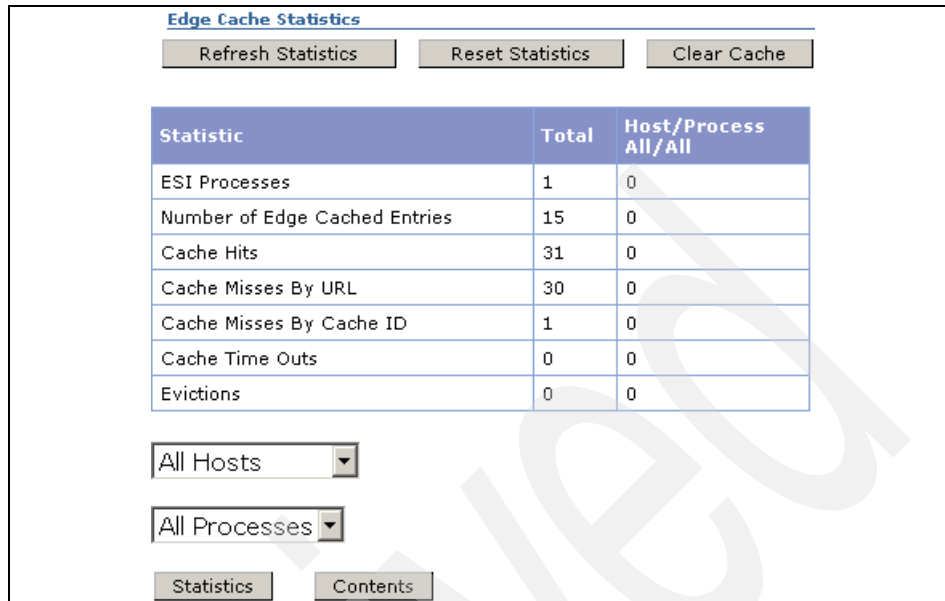


Figure 10-46 ESI statistics (Edge statistics)

The following information is available:

- **ESI Processes:** This is the number of processes configured as edge caches.
- **Number of Edge Cached Entries:** This is the number of entries currently cached on all edge servers and processes.
- **Cache Hits:** This is the number of requests that match entries on edge servers.
- **Cache Misses By URL:** A cache policy does not exist on the edge server for the requested template.

Note that the initial ESI request for a template that has a cache policy on a WebSphere Application Server results in a miss. Every request for a template that does not have a cache policy on the WebSphere Application Server will result in a miss by URL on the Edge server.

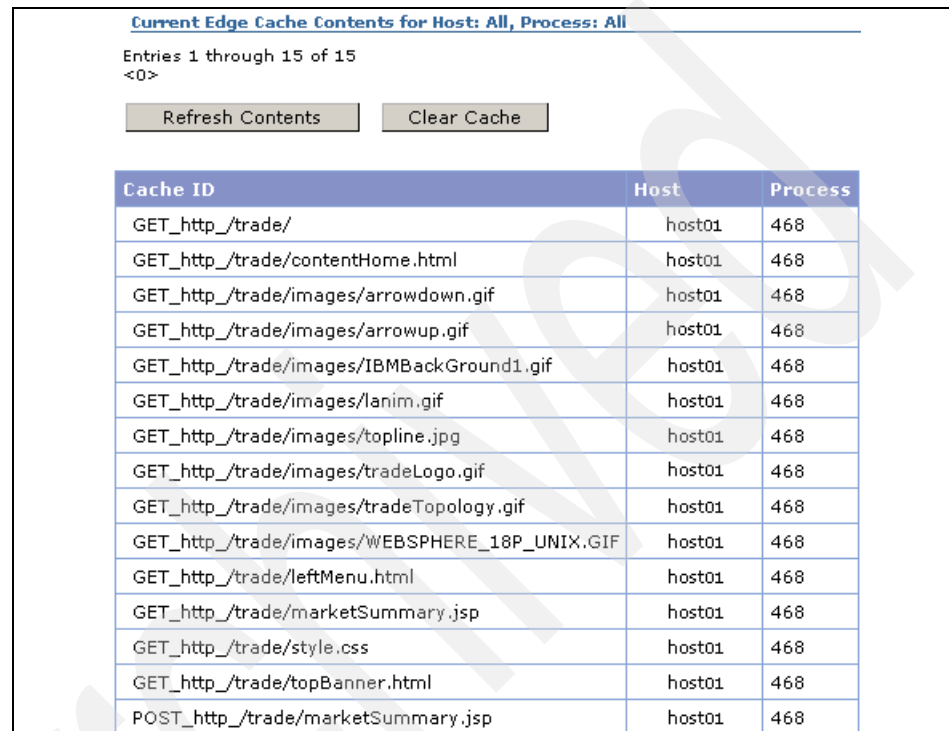
- **Cache Misses By Cache ID:** In this case, the cache policy for the requested template exists on the edge server. The cache ID is created (based on the ID rules and the request attributes), but the cache entry for this ID does not exist.

Note, that if the policy exists on the edge server for the requested template, but a cache ID match is not found, the request is not treated as a cache miss.

- **Cache Time Outs:** The number of entries removed from the edge cache, based on the timeout value.

- **Evictions:** The number of entries removed from the edge cache, due to invalidations received from WebSphere Application Server.

If you click the **Contents** button, you can see the edge content for all processes or for a selected process number. You can see an example in Figure 10-47.



Current Edge Cache Contents for Host: All, Process: All

Entries 1 through 15 of 15
<0>

Cache ID	Host	Process
GET_http_/trade/	host01	468
GET_http_/trade/contentHome.html	host01	468
GET_http_/trade/images/arrowdown.gif	host01	468
GET_http_/trade/images/arrowup.gif	host01	468
GET_http_/trade/images/IBMBackGround1.gif	host01	468
GET_http_/trade/images/lanim.gif	host01	468
GET_http_/trade/images/topline.jpg	host01	468
GET_http_/trade/images/tradeLogo.gif	host01	468
GET_http_/trade/images/tradeTopology.gif	host01	468
GET_http_/trade/images/WEBSPHERE_18P_UNIX.GIF	host01	468
GET_http_/trade/leftMenu.html	host01	468
GET_http_/trade/marketSummary.jsp	host01	468
GET_http_/trade/style.css	host01	468
GET_http_/trade/topBanner.html	host01	468
POST_http_/trade/marketSummary.jsp	host01	468

Figure 10-47 Edge cache content

The **Cache Policies** link shows the current cache policies, so you can make sure the cachespec.xml configuration is correct. The cache policies are always available in the Cache Monitor application, this information is not dependent on ESI caching.

Current Cache Policies			
Template	Class	Replication	Properties
/trade/app	servlet	Pull	
/trade/marketSummary.jsp	servlet	Pull	edgeable = true
/trade/register.jsp	servlet	Pull	edgeable = true
com.ibm.websphere.samples.trade.command.MarketSummaryCommand.class	command	None	
com.ibm.websphere.samples.trade.command.AccountCommand.class	command	None	
com.ibm.websphere.samples.trade.command.OrderCompletedCommand.class	command	None	
com.ibm.websphere.samples.trade.command.HoldingsCommand.class	command	None	

Figure 10-48 Current cache policies pane

Cache invalidation

There are three methods by which entries are removed from the ESI cache:

1. An entry's expiration timeout could fire.
2. An entry may be purged to make room for newer entries.
3. The application server could send an explicit invalidation for a group of entries.

10.6.3 External caching on the IBM HTTP Server

Restriction: The Fast Response Cache Accelerator (FRCA) is available for both Windows NT and Windows 2000 operating systems and AIX platforms. However, enabling FRCA cache for caching servlets and JSP files is only available on the Windows operating systems.

WebSphere Application Server fragment cache can use the IBM HTTP Server as an external cache. We explain the appropriate configuration in this section.

The FRCA cache resides on the Web server node as shown in Figure 10-49 on page 575.

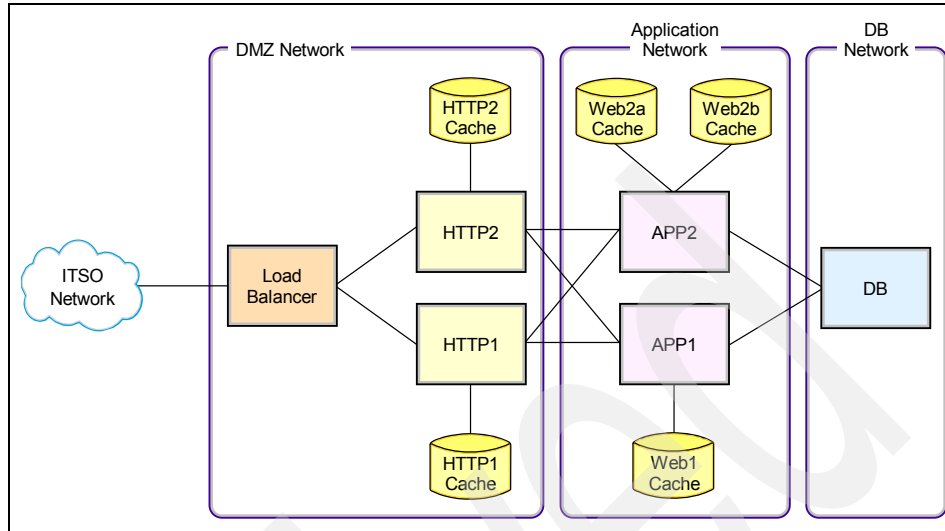


Figure 10-49 ITSO IBM HTTP Server infrastructure

Configuration steps

1. Configure the external cache group on the application servers (in our case Web1, Web2a, and Web2b). Follow the steps outlined in “WebSphere External Cache configuration” on page 560 and create an external cache group called “AFPA”. Then add a member with these parameters:

- Address: port
- Adapter bean name: `com.ibm.ws.cache.servlet.Afpa`

The port that you need to provide in the Address field is a port number that is available in the application server machine that will be used for communication between the AFPA component and the application server. See Figure 10-50 on page 576.

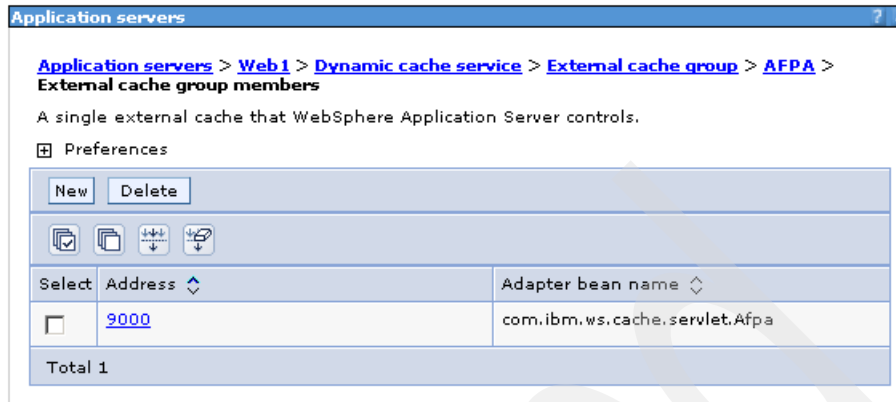


Figure 10-50 APFA external cache group member

Repeat this step for each application server. Make sure that you do not use the same port when the application servers are running in the same machine.

In our scenario, we used the following ports:

- Web1: port 9000
- Web2a: port 9000
- Web2b: port 9001

2. Add a cache policy to the cachespec.xml file for the servlet or JSP file you want to cache. In our example we added


```
<property name="ExternalCache">AFPA</property>
```

 for all entries that should be cached by FRCA.

Example 10-21 Cache entry for marketSummary.jsp

```
<cache-entry>
  <class>servlet</class>
  <name>/marketSummary.jsp</name>
  <property name="ExternalCache">AFPA</property>
  <cache-id>
    <priority>3</priority>
    <timeout>180</timeout>
  </cache-id>
  <dependency-id>MarketSummary
</dependency-id>
</cache-entry>
```

3. Update the Trade 6 application with the new cachespec.xml (refer to "Distributing the cachespec.xml file" on page 536).

4. Edit the IBM HTTP Server configuration file. You can use the WebSphere Administrative Console. Click **Servers -> Web servers -> <WebServer_Name> -> Configuration File.**

By default, the httpd.conf file already has an entry to load the AFPA module, but it is commented out, as shown in Example 10-22.

Example 10-22 Original AFPA configuration in the httpd.conf file

```
#LoadModule ibm_afpa_module modules/mod_afpa_cache.so
<IfModule mod_afpa_cache.c>
    Afpable
    Afpacache on
    Afpaport 80
    Afpalogfile "C:/WebSphere/IBM HTTP Server/logs/afpalog" V-ECLF
</IfModule>
```

Locate the lines shown in Example 10-22 and edit them according to the configuration shown in Example 10-23. Remove the `/IfModule` lines. Restart the Web servers after editing the configuration file.

Example 10-23 AFPA adapter configuration in httpd.conf file

```
LoadModule ibm_afpa_module modules/mod_afpa_cache.so
LoadModule afpapugin_20_module "C:/WebSphere/Plugins/bin/afpapugin_20.dll"
Afpable
Afpacache on
Afpaport 80
Afpalogfile "C:/IHS/logs/afpalog" V-ECLF
Afpapluginhost app1:9000
Afpapluginhost app2:9000
Afpapluginhost app2:9001
```

`Afpapluginhost WAS_Hostname:port` specifies the host name and port of the application server. These are the same values that you have specified in the Address field when configuring the external cache group member (step 1 on page 575).

The `LoadModule` directive loads the IBM HTTP Server plug-in that connects the Fast Response Cache Accelerator to the WebSphere Application Server fragment cache.

- If multiple IBM HTTP Servers are routing requests to a single application server, add the directives from Example 10-23 to the httpd.conf file of each of these IBM HTTP Servers.
- If one IBM HTTP Server is routing requests to a cluster of application servers, add the `Afpapluginhost WAS_Hostname:port` directive to the httpd.conf file for each application server in the cluster. For example, if there are three application servers in the cluster (Web1, Web2a, Web2b)

add an `AfpaPluginHost` directive for each application server as shown in Example 10-24). The ports are the same that we configured in step 1 on page 575.

Example 10-24 AFPA adapter configuration in httpd.conf file for application server cluster

```
AfpaPluginHost app1:9000
AfpaPluginHost app2:9000
AfpaPluginHost app2:9001
```

Do not forget to restart the Web servers after the `httpd.conf` file is changed.

Cache monitoring

You can monitor the FRCA cache log file. The location of this log file is also configured in the `httpd.conf` file (`AfpaLogFile` directive - see Example 10-23 on page 577).

10.6.4 External caching on the Caching Proxy

The dynamic caching function enables the Caching Proxy to cache dynamically generated content in the form of responses from JSPs and servlets generated by IBM WebSphere Application Server. A Caching Proxy adapter module is used at the application server to modify the responses, so that they can be cached at the proxy server in addition to being cached in the application server's dynamic cache. With this feature, dynamically generated content can be cached at the entry point to the network, avoiding repeated requests to the application server, when the same content is requested by multiple clients.

However, the Caching Proxy can only cache full pages, not fragments, and all subcomponents of that page must also be cacheable. Also, secure content requiring authorization is not cached externally at the proxy server.

Attention: Please notice that we found only one page in Trade 6 that adheres to this rule. This page is `register.jsp`. Therefore, it does not make much sense to use this dynamic caching scenario with Trade 6.

However, your application might be totally different and could benefit greatly from caching dynamic pages in the Caching Proxy.

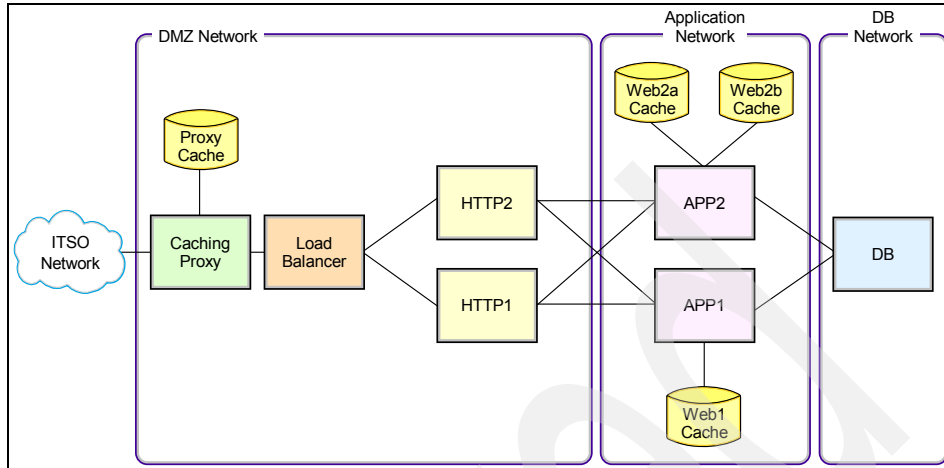


Figure 10-51 ITSO Caching Proxy infrastructure

The caches on the Caching Proxy and the WebSphere Application Server(s) are synchronized when a new result is generated or a cache object expires. Cached dynamic pages do not expire in the same way that regular files do. They can be invalidated by the application server that generated them.

Configuration steps

1. Configure the external cache group on the application servers (in our case Web1, Web2a and Web2b). Follow the steps outlined in “WebSphere External Cache configuration” on page 560 and create an external cache group. Then add a member with these parameters:

- Address: hostname:port
- Adapter bean name: `com.ibm.websphere.edge.dynacache.WteAdapter`

We have created a cache group called ITSO-APPSERV (Figure 10-42 on page 565) and added our Caching Proxy server to this group (see Figure 10-52 on page 580).

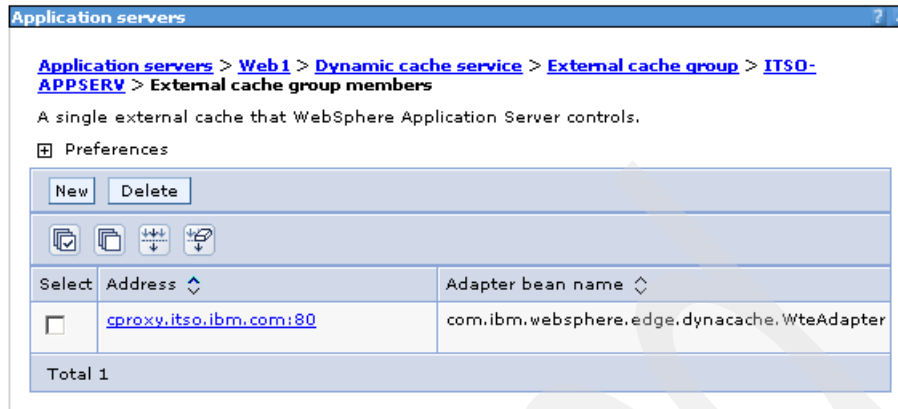


Figure 10-52 ITS0-APPSERV external cache group member setting

2. Edit the dynaedge-cfg.xml file, and add the external group statement as shown in Example 10-25. There is a sample provided with the product in the <WebSphere_install_root>/properties directory, but this is not used by the application servers. Make sure you copy the sample or create a new dynaedge-cfg.xml file in the <install_root>/profiles/<profile_name>/properties directory of each node. The user and userPasswd correspond to the administrator user of the Caching Proxy as defined in 5.7.1, "Using the Caching Proxy configuration wizard" on page 213.

Example 10-25 dynaedge-cfg.xml file

```
<?xml version="1.0"?>
<EdgeServerCfg CacheManager="ITS0-APPSERV">
  <EdgeServer
    endpoint = "http://cproxy.itso.ibm.com:80"
    user = "admin"
    userPasswd = "admin"
    invalidation-url = "/WES_External_Adapter"
    URI-type= "absolute"
  />
</EdgeServerCfg>
```

3. Add a cache policy to the cachespec.xml file for the servlet or JSP file you want to cache. Add <property name="ExternalCache">ITS0-APPSERV</property> for all entries which should be cached by the proxy cache server.

Example 10-26 Cache entry for register.jsp

```
<cache-entry>
  <class>servlet</class>
  <name>/register.jsp</name>
```

```
<property name="EdgeCacheable">true</property>
<property name="ExternalCache">ITS0-APPSERV</property>
<cache-id>
  <timeout>180</timeout>
</cache-id>
</cache-entry>
```

4. Configure dynamic caching at the Caching Proxy server.

The configuration changes are done in the caching proxy configuration file `ibmproxy.conf`. This file is stored in the `< caching_proxy_install_root >/etc/en_US` directory.

- Set the `Service` directive to enable the dynamic caching plug-in as shown in Example 10-27. The directive is already in the file, but it is commented out so simply remove the comment indicator. Note that each directive must appear on a single line in the proxy configuration file.

Example 10-27 Service directive

```
# ===== JSP Plug-in =====
Service /WES_External_Adapter
/opt/ibm/edge/cp/lib/plugins/dynacache/libdyna_plugin.o:exec_dynacmd
```

- Set the `ExternalCacheManager` directive to specify file sources. Each Caching Proxy must also be configured to recognize the source of the dynamically generated files. See Example 10-28. You need to add an `ExternalCacheManager` directive to the `ibmproxy.conf` file for each application server that caches dynamically generated content at this proxy server. This directive specifies a WebSphere Application Server or the cluster of application servers that caches results at the proxy, and sets a maximum expiration time for content from that server/cluster.

Example 10-28 ExternalCacheManager directive

```
#ExternalCacheManager directive:
ExternalCacheManager ITS0-APPSERV 20 minutes
```

- Next you need to add a `CacheQueries` directive to enable cache responses for requests that contain a question mark (?), for example `/trade/app?action=home`. See Example 10-29.

Example 10-29 CacheQueries directive

```
#CacheQueries directive:
CacheQueries ALWAYS http://cproxy.itso.ibm.com/*
```

- Set the `CacheTimeMargin`, which specifies the minimum expiration time; files with expiration times below this minimum are not cached. Because

query responses sometimes have very short expiration times, setting this directive to a lower setting allows more query responses to be cached. See Example 10-30.

Example 10-30 CacheTimeMargin directive

```
# CacheTimeMargin directive:
CacheTimeMargin 1 minutes
```

5. Restart the Caching Proxy and application servers.

After restarting the application servers, check the log file `<install_root>/profiles/<profile_name>/logs/edge/logs/dynaedge-<date>.log` to verify that your dynamic cache adapter was initialized properly. See Example 10-31.

Example 10-31 Dynamic cache adapter log entry

```
Nov 2, 2004 1:58:09 PM: Using file
/usr/WebSphere/AppServer/profiles/app2/properties/dynaedge-cfg.xml to initialize
Edge Server configurations.
```

For more information about configuring dynamic caching in the Caching Proxy, refer to the *Caching Proxy Administration Guide Version 6.0*, GC31-6857.

Caching Proxy monitoring

There is no GUI interface available for monitoring the cache content on the Caching Proxy server. Therefore, you need to look at some log files to understand what is cached and what is not.

The filenames and location are specified in the `ibmproxy.conf` file. The default location is the `<caching_proxy_install_root>/server_root/logs/` directory. The relevant log files are:

► Proxy access log

It is used for logging proxy requests. It is defined by the directive `ProxyAccessLog` in `ibmproxy.conf`.

The default filename is `proxy.<date_stamp>.<sequence>`.

► Cache access log

It is used for logging hits on proxy cache. It is defined by the directive `CacheAccessLog` in `ibmproxy.conf`.

The default filename is `cache.<date_stamp>.<sequence>`.

During our tests, we monitored these files in order to identify the components that were sent to the back-end servers and the components that were delivered from the cache.

For our first access to the Trade 6 application through the Caching Proxy server we accessed the main page, clicked the **Go Trade!** link and selected the **Register with Trade** link. This resulted in the following entries in the proxy log file:

Example 10-32 Entries in the proxy log file

```
10.20.10.151 - - [05/Nov/2004:12:29:25 -0500] "GET /trade/ HTTP/1.1" 200 688
10.20.10.151 - - [05/Nov/2004:12:29:25 -0500] "GET /trade/leftMenu.html
HTTP/1.1" 200 1501
10.20.10.151 - - [05/Nov/2004:12:29:25 -0500] "GET /trade/contentHome.html
HTTP/1.1" 200 2534
10.20.10.151 - - [05/Nov/2004:12:29:25 -0500] "GET /trade/style.css HTTP/1.1"
200 98
10.20.10.151 - - [05/Nov/2004:12:29:25 -0500] "GET
/trade/images/tradeTopology.gif HTTP/1.1" 200 30524
10.20.10.151 - - [05/Nov/2004:12:29:25 -0500] "GET /trade/images/tradeLogo.gif
HTTP/1.1" 200 3346
10.20.10.151 - - [05/Nov/2004:12:29:25 -0500] "GET
/trade/images/WEBSPHERE_18P_UNIX.GIF HTTP/1.1" 200 596
10.20.10.151 - - [05/Nov/2004:12:29:26 -0500] "GET /trade/topBanner.html
HTTP/1.1" 200 1484
10.20.10.151 - - [05/Nov/2004:12:29:26 -0500] "GET /trade/images/lanim.gif
HTTP/1.1" 200 1262
10.20.10.151 - - [05/Nov/2004:12:29:26 -0500] "GET /trade/images/topline.jpg
HTTP/1.1" 200 645
10.20.10.151 - - [05/Nov/2004:12:29:26 -0500] "GET
/trade/images/IBMBackGround1.gif HTTP/1.1" 200 43
10.20.10.151 - - [05/Nov/2004:12:29:51 -0500] "GET /trade/app HTTP/1.1" 200
2935
10.20.10.151 - - [05/Nov/2004:12:29:52 -0500] "GET /trade/register.jsp
HTTP/1.1" 200 4115
```

When we accessed those links for the second time, most entries were added to the cache log file, indicating that they were delivered from the cache. This is shown in Example 10-33.

Example 10-33 Entries in the cache log file

```
10.20.10.151 - - [05/Nov/2004:12:29:26 -0500] "GET /trade/images/tradeLogo.gif
HTTP/1.1" 304 0
10.20.10.151 - - [05/Nov/2004:12:29:26 -0500] "GET
/trade/images/WEBSPHERE_18P_UNIX.GIF HTTP/1.1" 304 0
10.20.10.151 - - [05/Nov/2004:12:29:51 -0500] "GET /trade/style.css HTTP/1.1"
304 0
```

```

10.20.10.151 - - [05/Nov/2004:12:30:57 -0500] "GET /trade/ HTTP/1.1" 304 0
10.20.10.151 - - [05/Nov/2004:12:30:57 -0500] "GET /trade/topBanner.html
HTTP/1.1" 304 0
10.20.10.151 - - [05/Nov/2004:12:30:57 -0500] "GET /trade/leftMenu.html
HTTP/1.1" 304 0
10.20.10.151 - - [05/Nov/2004:12:30:57 -0500] "GET /trade/contentHome.html
HTTP/1.1" 304 0
10.20.10.151 - - [05/Nov/2004:12:30:57 -0500] "GET /trade/images/lanim.gif
HTTP/1.1" 304 0
10.20.10.151 - - [05/Nov/2004:12:30:57 -0500] "GET /trade/images/topline.jpg
HTTP/1.1" 304 0
10.20.10.151 - - [05/Nov/2004:12:30:57 -0500] "GET
/trade/images/tradeTopology.gif HTTP/1.1" 304 0
10.20.10.151 - - [05/Nov/2004:12:30:57 -0500] "GET /trade/images/tradeLogo.gif
HTTP/1.1" 304 0
10.20.10.151 - - [05/Nov/2004:12:30:57 -0500] "GET
/trade/images/WEBSPHERE_18P_UNIX.GIF HTTP/1.1" 304 0
10.20.10.151 - - [05/Nov/2004:12:30:57 -0500] "GET /trade/style.css HTTP/1.1"
304 0
10.20.10.151 - - [05/Nov/2004:12:30:57 -0500] "GET
/trade/images/IBMBackGround1.gif HTTP/1.1" 304 0
10.20.10.151 - - [05/Nov/2004:12:30:57 -0500] "GET /trade/images/tradeLogo.gif
HTTP/1.1" 304 0
10.20.10.151 - - [05/Nov/2004:12:30:58 -0500] "GET
/trade/images/WEBSPHERE_18P_UNIX.GIF HTTP/1.1" 304 0
10.20.10.151 - - [05/Nov/2004:12:30:59 -0500] "GET /trade/style.css HTTP/1.1"
304 0
10.20.10.151 - - [05/Nov/2004:12:31:01 -0500] "GET /trade/register.jsp
HTTP/1.1" 304 0

```

Deleting Caching Proxy cache content

You have two options to delete the Caching Proxy cache:

- ▶ If you are running with a memory cache only, then the only way to purge the cache is to restart the process.
- ▶ If you are running with a raw disk cache, then you can use the **htcformat** command to reformat the cache device.

Caching Proxy troubleshooting

You can enable tracing for the Caching Proxy. To do this, you need to specify the log file for tracing in the `ibmproxy.conf` file as shown in Example 10-34.

Example 10-34 Tracing log file

```

#log file directives
TraceLog /opt/ibm/edge/cp/server_root/logs/trace

```

Also, you can enable tracing on the application server for `com.ibm.websphere.edge.*`, which is the package containing the external cache adapters. To do this, click **Troubleshooting -> Logs and Trace -> <AppServer_Name> -> Diagnostic Trace** in the WebSphere Administrative Console. Click the **Configuration** tab, click the **Enable Log** checkbox and click **OK**.

Click **Troubleshooting -> Logs and Trace -> <AppServer_Name> -> Change Log Detail Levels**, click the **Configuration** tab and enter the trace string `com.ibm.websphere.edge.*=all`. Click **Apply** or **OK** and restart your application server(s).

Note that `*=info` is the default value, so we recommend keeping it in the configuration.

An example of this window is shown in Figure 10-53.



Figure 10-53 Setting up the trace string for Caching Proxy

External cache invalidation

The responsibility for synchronizing the dynamic cache of external caches and the WebSphere Application Server is shared by both systems. For example, a public Web page dynamically created and cached at the application server using Servlet/JSP result caching can be exported by the application server and cached by the external cache server, which can serve the application's execution results repeatedly to many different users until notified that the page is invalid.

The content in the external cache is valid until:

- ▶ The proxy server removes an entry because the cache is congested.
- ▶ Or the default timeout set by the Caching Proxy's configuration file expires.
- ▶ Or the Caching Proxy receives an Invalidate message directing it to purge the content from its cache. Invalidate messages originate at the WebSphere Application Server that owns the content and are propagated to each configured Caching Proxy.

10.7 Using the Dynamic Cache Policy Editor

As mentioned earlier, the Dynamic Cache Policy Editor is provided as an Eclipse plug-in and plugs into WebSphere Studio Application Developer V5.1.0 and higher or Application Server Toolkit (AST) V5.1.0 and higher and can be downloaded from:

<http://www.alphaworks.ibm.com/tech/cachepolicyeditor>

Restriction: At the time of writing this redbook, the Dynamic Cache Policy Editor could not be used with Rational Application Developer V6 or AST V6.

As it is not expected that the GUI changes much once available for Rational Application Developer V6 or AST V6, we decided to include this section to show the functionality of the tool. All screenshots and descriptions are based on the available Dynamic Cache Policy Editor version plugged into AST V5.1.

Please check the download site for new versions of the product.

10.7.1 Dynamic Cache Policy Editor installation

The installation procedure for the Dynamic Cache Policy Editor is similar to any Eclipse plug-in installation. We used AST for our examples but the same procedure applies for WebSphere Studio Application Developer and should also be identical for Rational Application Developer once supported.

1. Close AST before continuing.
2. Open the file you downloaded from the alphaWorks® site (com.ibm.etools.cpe_5.1.0.1.zip) with a zip extraction software and extract it to the directory <tool_install_path>\eclipse\plugins, where <tool_install_path> is the installation path of AST. In our scenario, we extracted it to C:\Program Files\IBM\WebSphere\AST\eclipse\plugins.

The files are extracted to a directory named com.ibm.etools.cpe_5.1.0.1, as shown in Figure 10-54 on page 587.

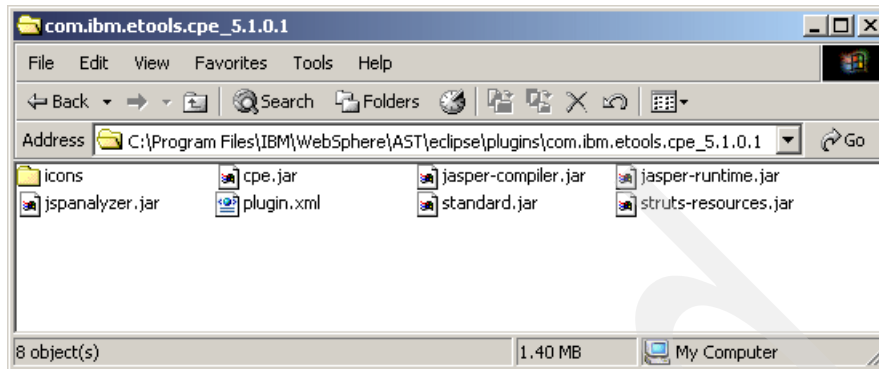


Figure 10-54 Directory containing the Dynamic Cache Policy Editor files

3. Open AST.
4. Locate a Web module inside your application, and select it. In the menu, click **File -> New -> Other...**
5. In the New window, select **Web** in the left pane. In the right pane you can see the new option that was added, which is **Cache Policy File**, as shown in Figure 10-55.

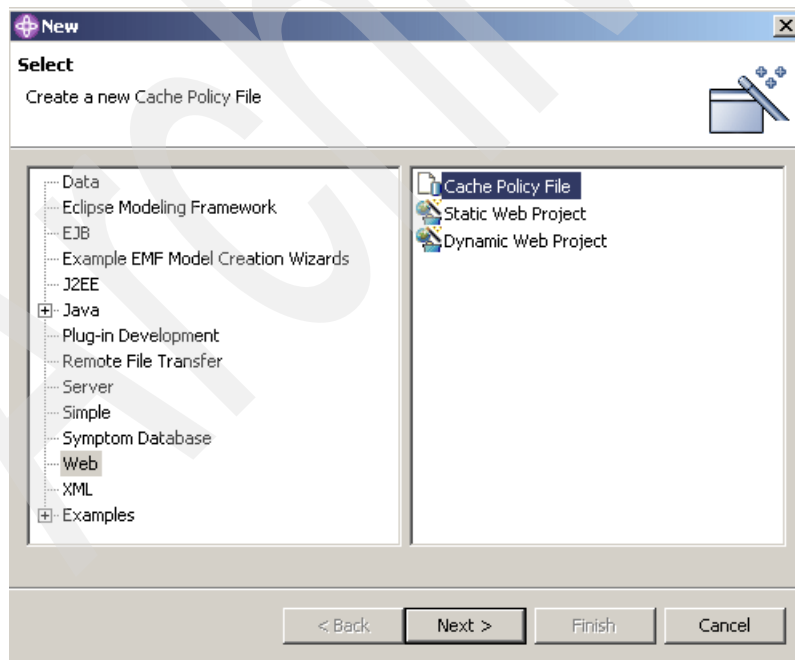


Figure 10-55 Cache Policy File option

6. In the New cache policy file window, you can select whether you want Dynamic Cache Policy Editor to create and populate the cachespec.xml file with policies for the servlets and JSPs in your application, or it can create an empty file, so you can populate it yourself. Select the proper option, and click **Finish**, as shown in Figure 10-56.

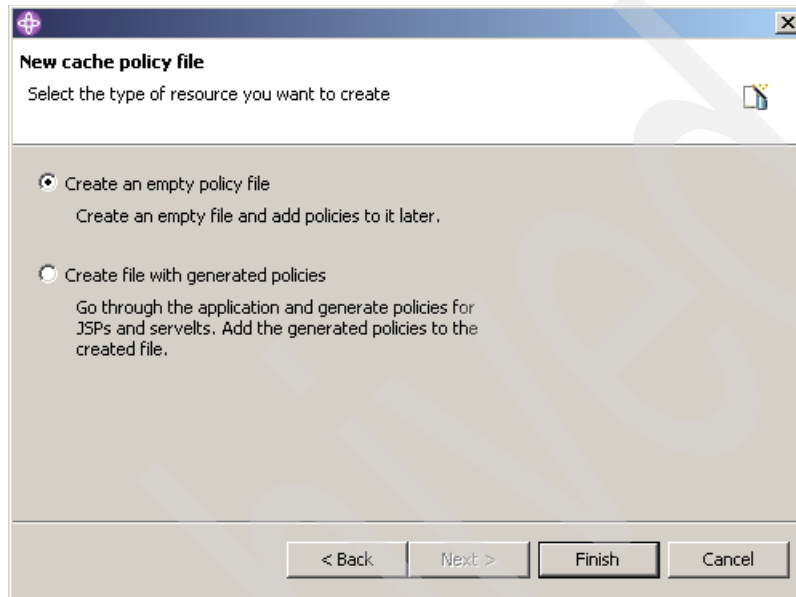


Figure 10-56 Creating a policy file

Note: If you select a component that is not a Web application and you try to create a cache policy file using the Dynamic Cache Policy Editor, you will see an error message informing that the WEB-INF directory was not found or, in some situations, the Finish button does not work. Therefore, make sure you select a Web application before trying to create the cache policy file.

Check for the new cache policy file inside the Web application, in the WEB-INF directory, as shown in Figure 10-57 on page 589.

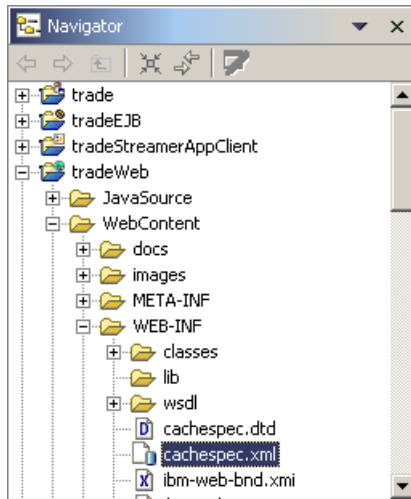


Figure 10-57 Cache policy file created by the wizard

The Cache Policy Editor is automatically opened after the file creation has finished. Whenever you need to open it, double-click the **cachespec.xml** file. The Cache Policy Editor is shown in Figure 10-58.

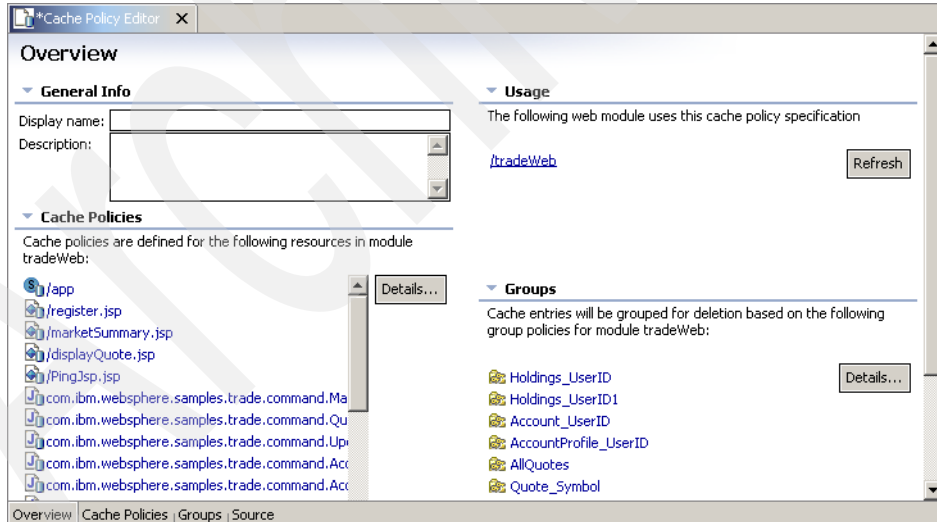


Figure 10-58 Cache Policy Editor overview pane

The Cache Policy Editor provides four pages: Overview (default), Cache Policies, Groups and Source. It provides wizards for most tasks, and the source editor provides content completion and content validation.

It also shows messages in the Workbench Tasks view and updates the Outline view, showing the structure of the cache policies file, as shown in Figure 10-59.

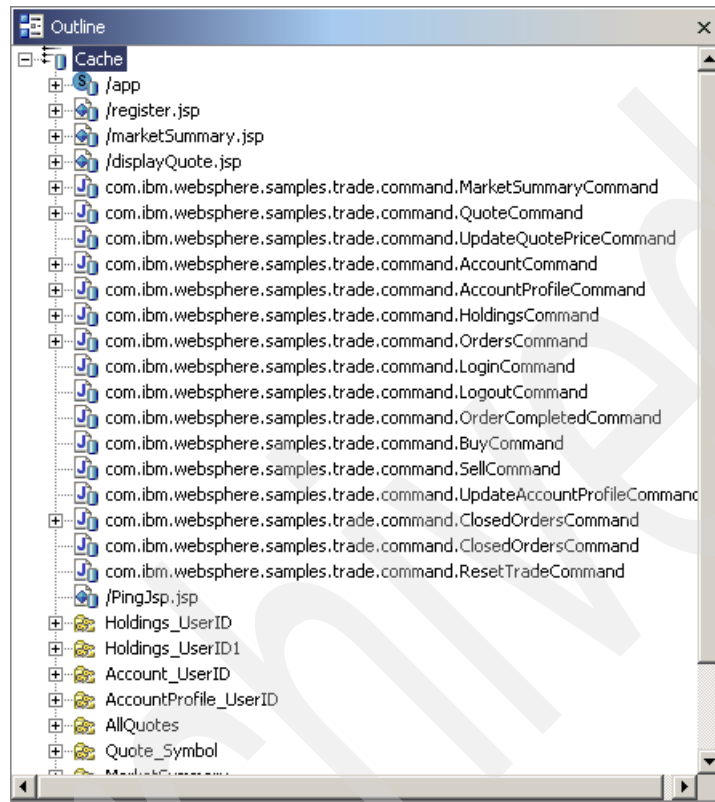


Figure 10-59 Outline view for the cache policies

10.7.2 Creating cache policy entries

If you choose to create an empty cache policy file, you can add elements to it by using context menu options of the Dynamic Cache Policy Editor.

You can create policies for all servlets and JSPs at once by right-clicking the Web application (**tradeWeb** in our example) and selecting **Generate Cache Policies**, as shown in Figure 10-60 on page 591.

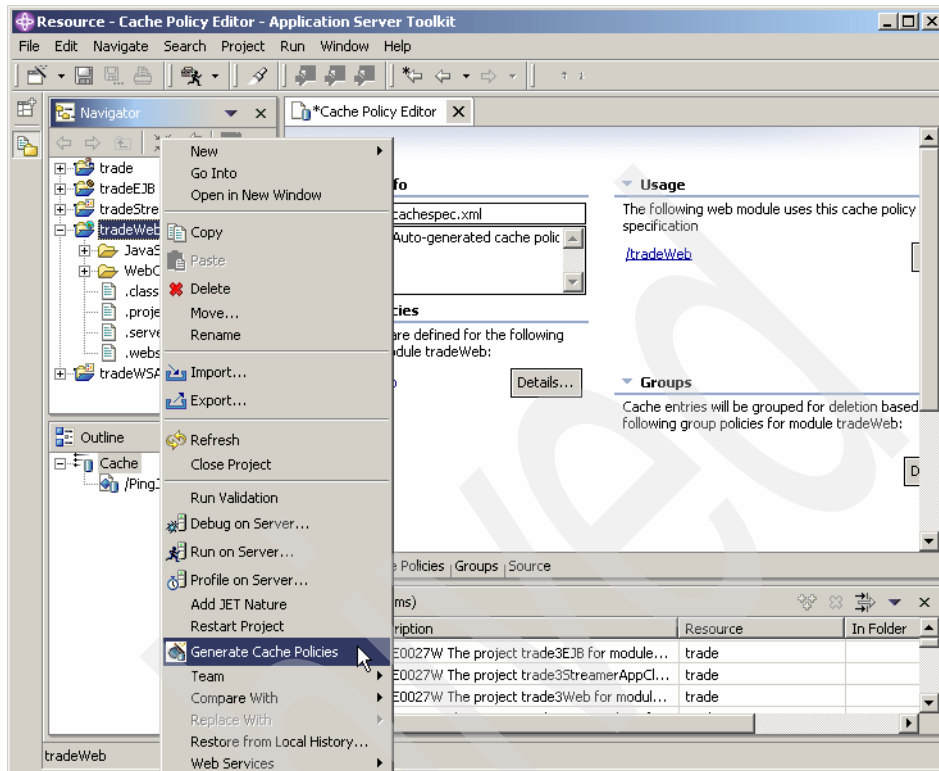


Figure 10-60 Generate Cache Policies menu option

You can also create policies for specific elements, by right-clicking them and selecting **Create Cache Policy**, as shown in Figure 10-61 on page 592.

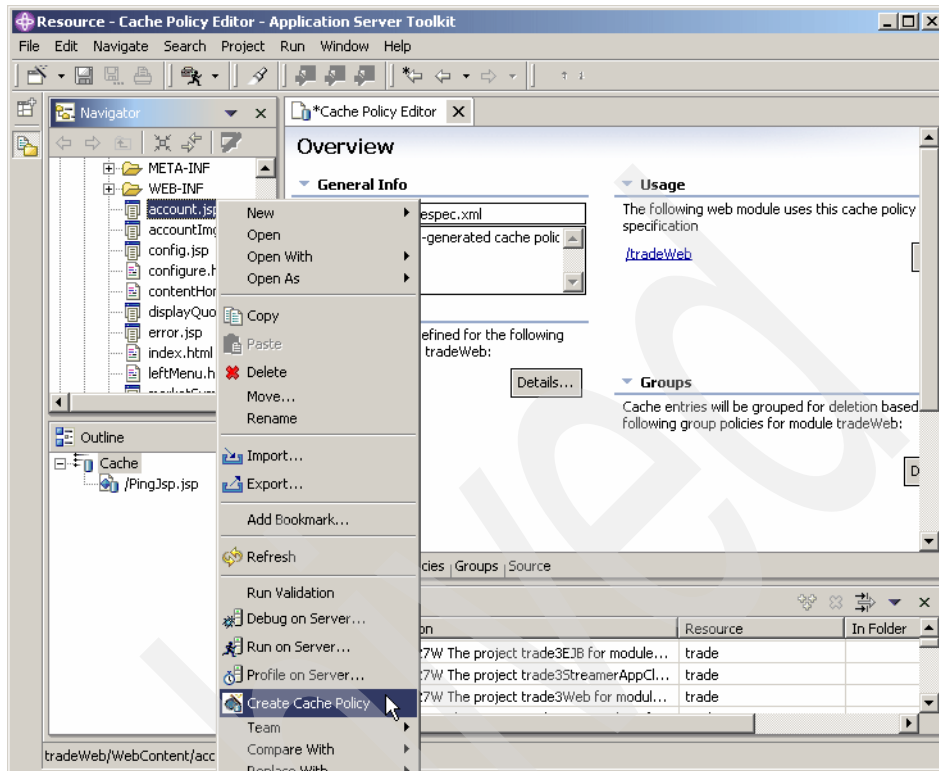


Figure 10-61 Creating a policy for a selected element

10.7.3 Examples: Creating cachespec.xml entries with the Dynamic Cache Policy Editor

In this section we explain how to create cachespec.xml entries for the home servlet, MarketSummaryCommand (which contains a dependency entry) and UpdateQuotePriceCommand (which contains an invalidation entry), as used in our examples in 10.5, “WebSphere dynamic caching scenarios” on page 528 and 10.6, “WebSphere external caching scenarios” on page 558.

For more information about the Dynamic Cache Policy Editor, refer to 10.3.2, “Dynamic Cache Policy Editor” on page 515.

Create an entry for home servlet

The following steps describe how to create an entry for the home servlet, similar to the one described in “Creating and editing Trade 6 cachespec.xml file” on page 533.

1. Follow the instructions on 10.7, “Using the Dynamic Cache Policy Editor” on page 586 to create a blank `cachespec.xml` file. Note that in our examples we use Application Server Toolkit (AST).
2. Double-click the file **cachespec.xml** to open the Cache Policy Editor.
3. In the Cache Policy Editor, click the **Cache Policies** tab at the bottom of the Cache Policy Editor pane.
4. In the Cache Policies pane, click the **Add Cache Policy** button, as shown in Figure 10-62.

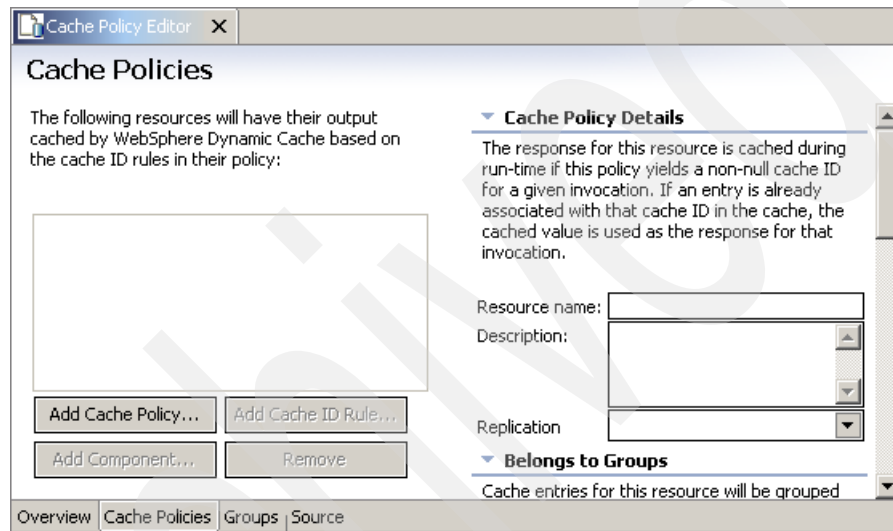
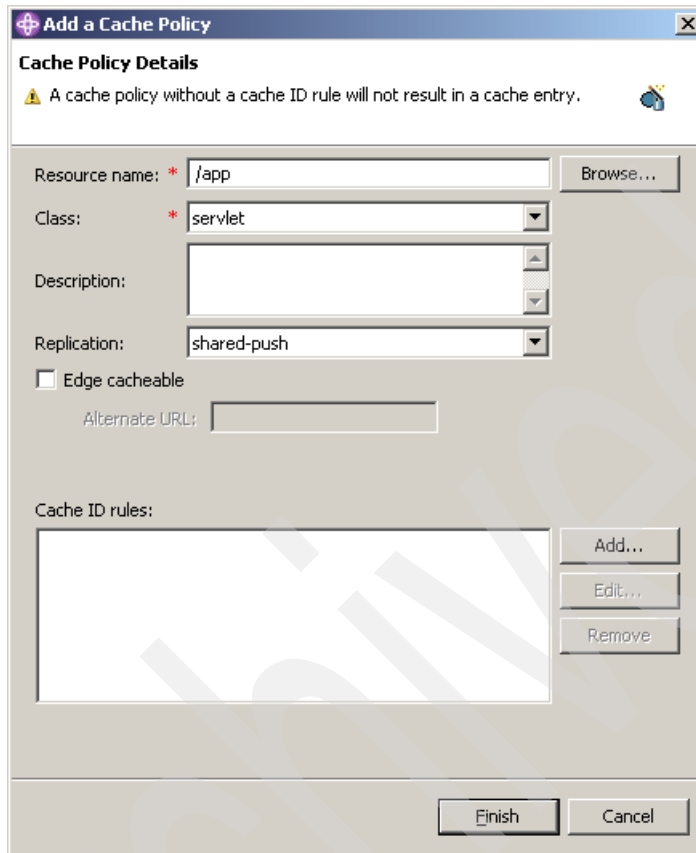


Figure 10-62 Cache Policies pane

5. In the Add a Cache Policy window, enter `/app` for the Resource name, select **servlet** in the Class field and select **shared-push** in the Replication field, as shown in Figure 10-63 on page 594.



The image shows a Windows-style dialog box titled "Add a Cache Policy". It has a standard title bar with a maximize button, a close button, and a help icon. Below the title bar is a section labeled "Cache Policy Details" with a warning icon and a message: "A cache policy without a cache ID rule will not result in a cache entry." The main area of the dialog contains several fields and controls: "Resource name:" with a text box containing "/app" and a "Browse..." button; "Class:" with a dropdown menu showing "servlet"; "Description:" with a text box and up/down arrow buttons; "Replication:" with a dropdown menu showing "shared-push"; an unchecked checkbox labeled "Edge cacheable"; and an "Alternate URL:" text box. At the bottom is a "Cache ID rules:" section with a large empty list box and three buttons: "Add...", "Edit...", and "Remove". At the very bottom of the dialog are "Finish" and "Cancel" buttons.

Figure 10-63 Adding a cache policy entry

6. Click the **Add...** button beside the Cache ID rules list box. In the Add a Cache ID rule window, enter Example1 in the Display name field, and click the **Add...** button beside the Components list box, as shown in Figure 10-64 on page 595.

Add a Cache ID Rule

Cache ID Details

⚠ A cache ID rule without a component will result in the same cache ID for every invocation of the resource.

Display Name:

Description:

Timeout:

Priority:

☐ Edge cacheable

Alternate URL:

Components:

Add...
Edit...
Remove

Finish Cancel

Figure 10-64 Adding a cache ID rule

7. In the Add a Component to a Cache ID Rule window, select **parameter** in the Type field, enter **action** in the Name/ID field, click the **Required** checkbox and click **Finish**.

Add a Component to a Cache ID Rule

Component Details
Create a component

Type: * parameter

Name / ID: * action

☒ Required
☐ Ignore Value

Method/Field

Values

☒ Use only these values ☐ Exclude these values

Finish Cancel

Figure 10-65 Adding the first component to the cache ID rule

8. Back to the Add a Cache ID rule window, click the button **Add...** again. In the Add a Component to a Cache ID Rule window, select **cookie** in the Type field, enter **JSESSIONID** in the Name/ID field, click the **Required** checkbox and click **Finish**.

Add a Component to a Cache ID Rule

Component Details
Create a component

Type: * cookie

Name / ID: * JSESSIONID

☒ Required
☐ Ignore Value

Method/Field

Values

☒ Use only these values ☐ Exclude these values

Finish Cancel

Figure 10-66 Adding the second component to the cache ID rule

9. Click **Finish** twice to close both windows.

An entry for the home servlet (/app) is added to the Cache Policies list box, as shown in Figure 10-67 on page 598.

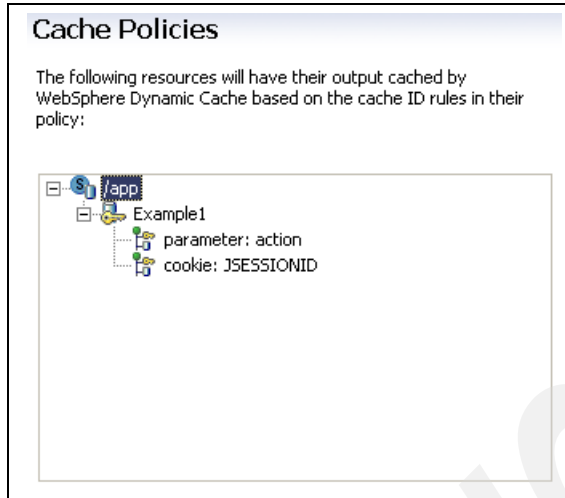


Figure 10-67 Cache policy for the home servlet

The XML source code generated by those steps is shown in Example 10-35.

Example 10-35 Entry created for the servlet home in the cachespec.xml file

```

<cache-entry>
  <class>
    servlet</class>
  <cache-id>
    <display-name>Example1</display-name>
    <component id="action" type="parameter" ignore-value="false">
      <required>
        true</required>
      </component>
    <component id="JSESSIONID" type="cookie" ignore-value="false">
      <required>
        true</required>
      </component>
    <timeout>
      0</timeout>
    <priority>
      1</priority>
    <property name="edgeCacheable">
      false</property>
    </cache-id>
  <name>
    /app</name>
  <sharing-policy>
    shared-push</sharing-policy>
  <property name="edgeCacheable">

```



```
false</property>
<property name="description">
</property>
<property name="externalCache">
</property>
<property name="alternate_url">
</property>
<property name="consume-subfragments">
</property>
<property name="persist-to-disk">
</property>
<property name="save-attributes">
</property>
<property name="store-cookies">
</property>
</cache-entry>
```

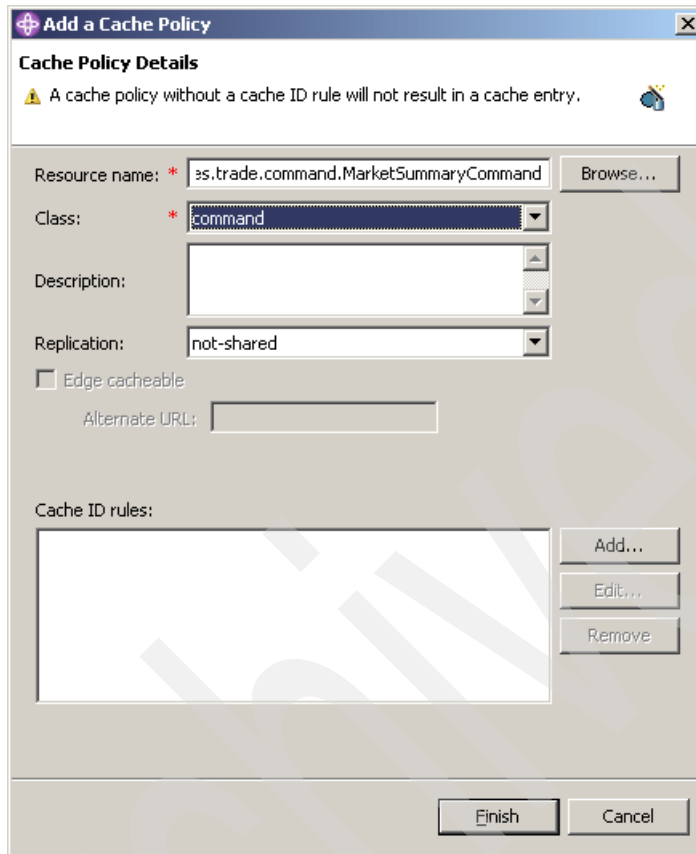
Important: When using the wizard, even if you do not select an option or do not enter a value into a field, default values are generated in the cachespec.xml.

Therefore, we recommend you check the final cachespec.xml file by clicking the Source pane in the Cache Policy Editor and eliminate any unwanted fields.

Create an entry for MarketSummaryCommand

The following example describes how to create a cache entry for MarketSummaryCommand. Refer to 10.5.3, “Command caching” on page 540 for more information.

1. In the Cache Policies pane of the Cache Policy Editor window, click the **Add Cache Policy** button (refer to steps 1 on page 593 through 4 on page 593).
2. In the Add a Cache Policy window, enter `com.ibm.websphere.samples.trade.command.MarketSummaryCommand` in the Resource name field, select **command** in the Class field and click the **Add...** button.



The image shows a Windows-style dialog box titled "Add a Cache Policy". It has a standard title bar with a maximize button, a close button, and a help icon. Below the title bar, the text "Cache Policy Details" is displayed. A warning icon and message state: "A cache policy without a cache ID rule will not result in a cache entry." The form contains several fields: "Resource name:" with a text box containing "es.trade.command.MarketSummaryCommand" and a "Browse..." button; "Class:" with a dropdown menu showing "command"; "Description:" with a text box and up/down arrow buttons; "Replication:" with a dropdown menu showing "not-shared"; an unchecked checkbox labeled "Edge cacheable"; and an "Alternate URL:" text box. At the bottom, there is a section for "Cache ID rules:" with a large empty text area and three buttons: "Add...", "Edit...", and "Remove". At the very bottom of the dialog are "Finish" and "Cancel" buttons.

Figure 10-68 Adding a cache policy for MarketSummaryCommand

3. Enter Example2 in the Display name field, enter 10 in the Timeout field and enter 3 in the priority field. Click **Finish**.

Add a Cache ID Rule

Cache ID Details

⚠ A cache ID rule without a component will result in the same cache ID for every invocation of the resource.

Display Name:

Description:

Timeout:

Priority:

☐ Edge cacheable

Alternate URL:

Components:

Add... Edit... Remove

Finish Cancel

Figure 10-69 Adding a new cache ID rule

4. Click **Finish** again to close the Add a Cache Policy window.
5. In order to create the dependency ID entry for this cache policy, select the Groups tab in the Cache Policy Editor Window and click the **Add Group...** button below the Groups pane.
6. Enter MarketSummary in the Name field, and click the **Add...** button beside the Members list box, as shown in Figure 10-70 on page 602.

Add a Group Policy

Group Details

⚠ Groups without members or invalidators will never be used.

Name: *

Description:

Members:

Add...
Edit...
Remove

Invalidators:

Add...
Edit...
Remove

Finish Cancel

Figure 10-70 Adding the MarketSummary group policy

7. In the Add a Membership Rule, select the marketSummaryCommand entry in the Policy field and click **Finish** twice.

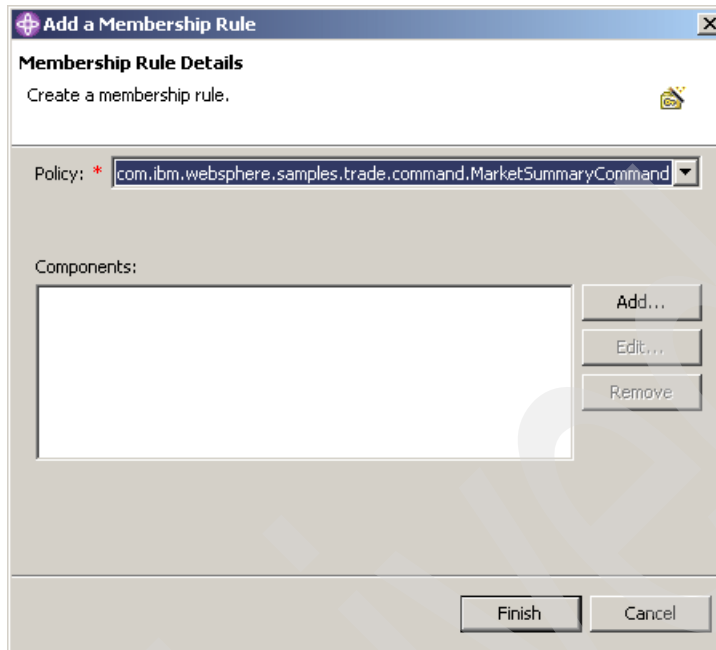


Figure 10-71 Adding a member of the MarketSummary group policy

The XML source code generated by those steps is shown in Example 10-36.

Example 10-36

```
<cache-entry>
  <class>
    command</class>
  <cache-id>
    <display-name>Example2</display-name>
    <timeout>
      10</timeout>
    <priority>
      3</priority>
    </cache-id>
  <name>
    com.ibm.websphere.samples.trade.command.MarketSummaryCommand</name>
  <sharing-policy>
    not-shared</sharing-policy>
  <dependency-id>
    MarketSummary</dependency-id>
</cache-entry>
<group name="MarketSummary">
  <description></description>
```

Create an entry for UpdateQuotePriceCommand

The following example describes how to create a cache entry for UpdateQuotePriceCommand, which contains an invalidation entry.

1. In the Cache Policies pane of the Cache Policy Editor window, click the **Add Cache Policy** button (refer to steps 1 on page 593 through 4 on page 593).
2. In the Add a Cache Policy window, enter `com.ibm.websphere.samples.trade.command.UpdateQuotePriceCommand` in the Resource name field, select **command** in the Class field and click the **Add...** button.
3. Enter `Example3` in the Display name field and click **Finish**.
4. Click **Finish** again to close the Add a Cache Policy window.
5. In order to create the invalidation entry for this cache policy, select the Groups tab in the Cache Policy Editor Window and click the **Add Group...** button below the Groups pane.
6. Enter `Quote_Symbol` in the Name field, and click the **Add...** button beside the Invalidators list box, as shown in Figure 10-72 on page 605.

Add a Group Policy

Group Details

⚠ Groups without members or invalidators will never be used.

Name: * Quote_Symbol

Description:

Members:

Invalidators:

Add... Edit... Remove

Add... Edit... Remove

Finish Cancel

Figure 10-72 Adding the Quote_Symbol group policy

7. In the Add an Invalidation Rule, select the UpdateQuotePriceCommand entry in the Policy field and click **Add...**, as shown in Figure 10-73 on page 606.

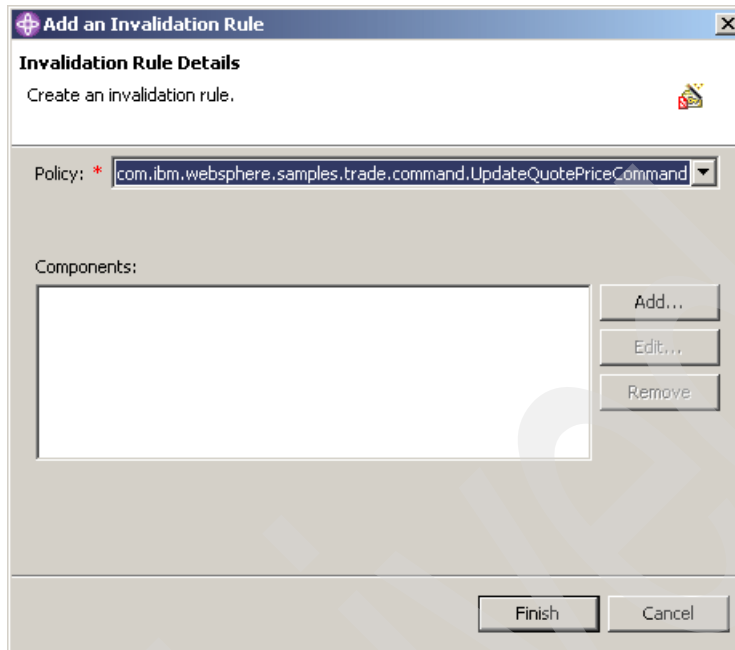


Figure 10-73 Adding an invalidation rule

8. In the Add a Component to a Group Invalidator Rule window, select **method** in the Type field, enter getSymbol in the ID field and click the **Required** checkbox.

Figure 10-74 Adding a component to the group invalidator rule

9. Click **Finish** three times.

The XML source code generated by those steps is shown in Example 10-36 on page 603.

Example 10-37

```
<cache-entry>
  <class>
    command</class>
  <cache-id>
    <display-name>Example3</display-name>
    <timeout>
      0</timeout>
    <priority>
```

```

        1</priority>
    </cache-id>
    <name>
com.ibm.websphere.samples.trade.command.UpdateQuotePriceCommand</name>
    <sharing-policy>
not-shared</sharing-policy>
    <property name="description">
    </property>
    <property name="externalCache">
    </property>
    <property name="alternate_url">
    </property>
    <property name="consume-subfragments">
    </property>
    <property name="edgeCacheable">
    </property>
    <property name="persist-to-disk">
    </property>
    <property name="save-attributes">
    </property>
    <property name="store-cookies">
    </property>
    <invalidation>
Quote_Symbol<component id="getSymbol" type="method"
ignore-value="false">
    <required>
true</required>
    </component></invalidation>
    </cache-entry>
    <group name="Quote_Symbol">
    <description></description>
    </group>

```

The Outline view shows the structure of our current cachespec.xml file, as shown in Figure 10-75 on page 609.

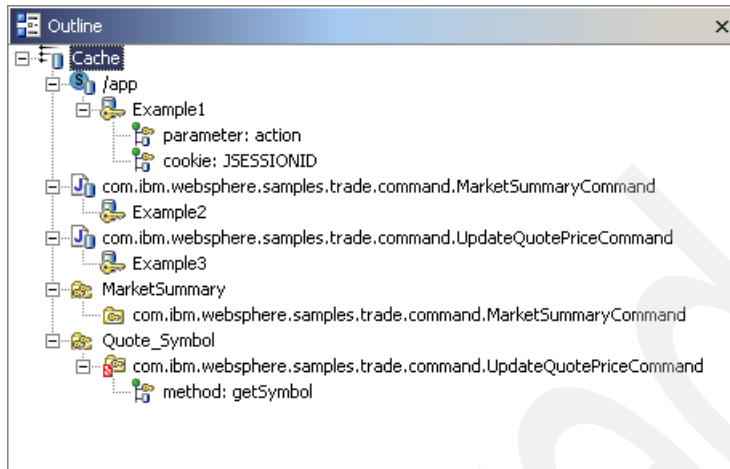


Figure 10-75 Outline view of the cachespec.xml file contents

10.8 Conclusion

As more e-business sites seek to retain customers by serving personalized content, they face potential server-side bottlenecks, slow user response time, and increasing infrastructure costs. The WebSphere Application Server dynamic cache service can solve these critical challenges. Caching dynamic content that needs back-end requests or CPU-intensive computations can reduce server-side bottlenecks and maximize system resources, thus boosting performance and reducing infrastructure costs.

The WebSphere Application Server dynamic cache service is easy to use and readily available. You can benefit from using the available comprehensive functions for caching dynamic content. The Servlet/JSP result cache and command cache make the caching of dynamic content possible - at various levels of granularity for the highest possible cache hits. The replication and invalidation support facilitates caches that can be shared, replicated, and synchronized in multi-tier or multi server environments. The Edge of Network Caching Support, with its external caches and fragment support, generates a virtual extension of application server caches into the network.

The WebSphere Application Server dynamic cache service combined with an appropriate external cache, such as WebSphere Edge Server or IBM HTTP Server, can power high volume sites with intensive dynamic content to achieve the highest level of scalability and performance.

The last information in this chapter is a short recapitulation of the caching technologies available for WebSphere Application Server solutions. Refer to Table 10-1.

Table 10-1 Caching techniques

	Middleware	What is cached	Memory or file system
Dynamic Caching	WebSphere Application Server	Results from JSPs Servlets	Both
Fast Response Cache Accelerator ^a	IBM HTTP Server	Results from JSP Servlets	V1.3 - Memory V2.0 - Both ^b
Web server plug-in	WebSphere Application Server - ESI processor	Results from JSP Servlets	Memory
Caching Proxy	WebSphere Edge Components	Results from JSP Servlets	Both

a. The Fast Response Cache Accelerator (FRCA) is available for both Windows NT and Windows 2000 operating systems and AIX platforms. However, enabling FRCA cache for caching servlets and JSP files is only available on the Windows operating systems.

b. IHS 2.0 provides a module called mod_mem_cache, which supports a memory cache and a file descriptor cache.

Despite the fact that we did not focus on static content caching, Table 10-2 provides an overview of static content caching technologies:

Table 10-2 Static caching

	Middleware	Memory or file system
Fast Response Cache Accelerator	IBM HTTP Server	V1.3 - Memory V2.0 - Both
Web server plug-in	WebSphere Application Server - ESI processor	Memory
Caching Proxy	WebSphere Edge Components	Both

10.9 Benchmarking Trade 3

This section is based on WebSphere V5.1 benchmarks using Trade 3 but it gives you a good impression of what performance benefits you can gain with dynamic caching. Please remember that all results depend on the underlying application - the more cacheable content, the bigger the performance gains.

There is no complete testing scenario available for WebSphere V6 and Trade 6 yet but the first benchmarks done with WebSphere V6 and Trade 6 have shown even better results than with the previous versions. One important thing to note is that DistributedMap caching performs better than command caching.

10.9.1 Dynamic caching

Note: This section is extracted from the existing testing scenario for dynamic caching of Trade 3.1.

During this test the system was stressed in these four scenarios:

- ▶ No caching (see Figure 10-76 on page 612).
- ▶ Enabled command caching (see Figure 10-77 on page 612).
- ▶ Enabled servlet and JSP result caching (see Figure 10-78 on page 613).
- ▶ Enabled Edge Side Include caching (see Figure 10-78 on page 613).

The test targeted both read and update operations as follows:

- ▶ Read (75%)
 - quote
 - portfolio
 - home
 - account
- ▶ Update (25%)
 - buy
 - sell
 - login/logout
 - register

As you can see in these figures, the command caching increased the performance 2.7 times, plus Servlet/JSP caching = 2.9 times and when edge caching was enabled, the performance was increased 4 times.

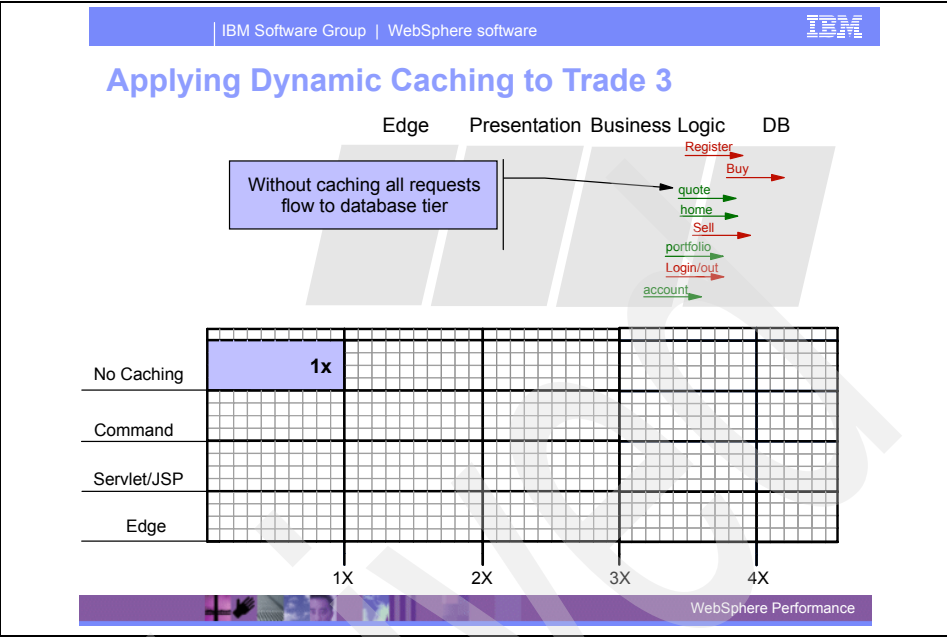


Figure 10-76 System performance without caching

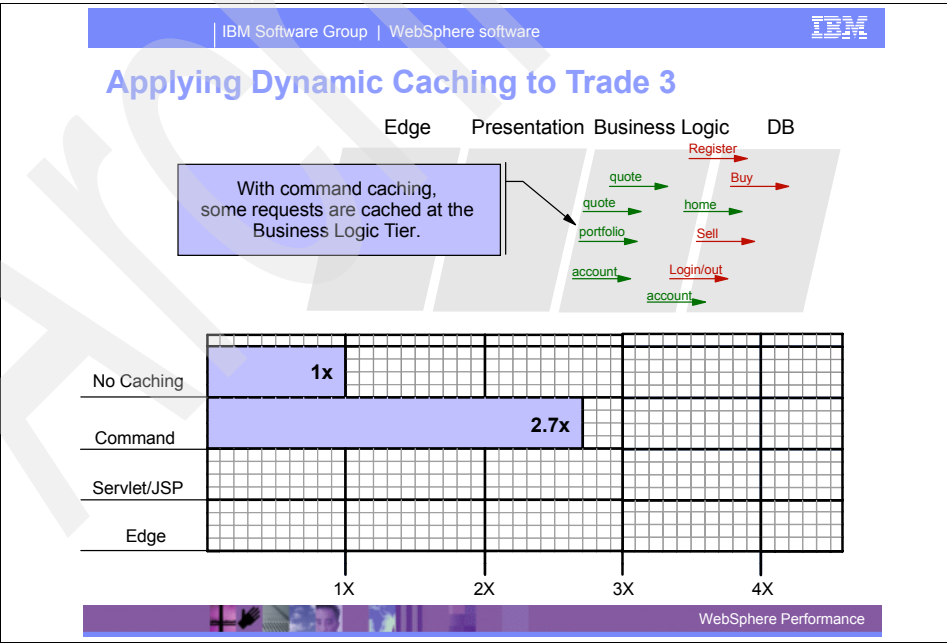


Figure 10-77 System performance with command caching

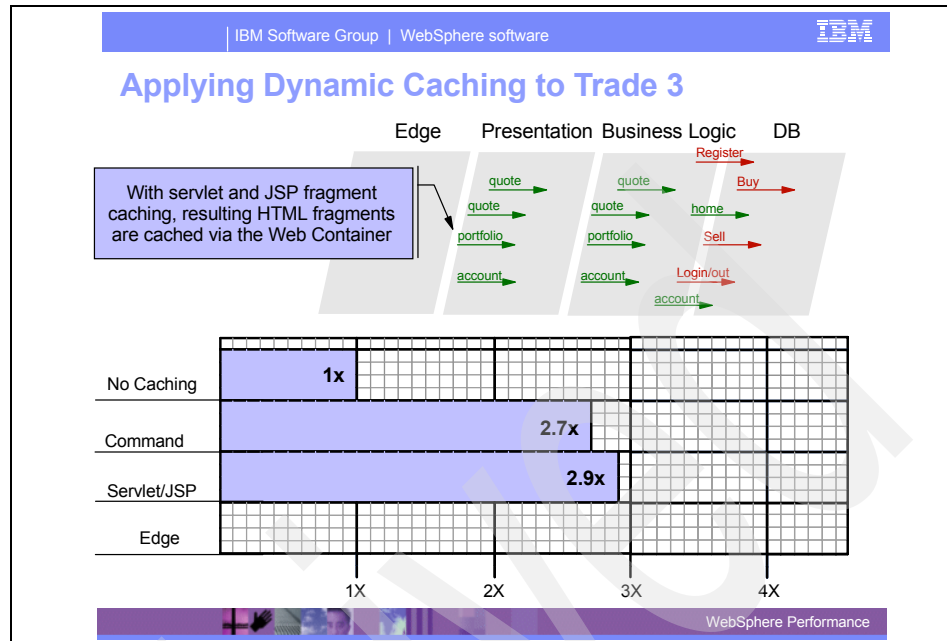


Figure 10-78 System performance with servlet and JSP caching

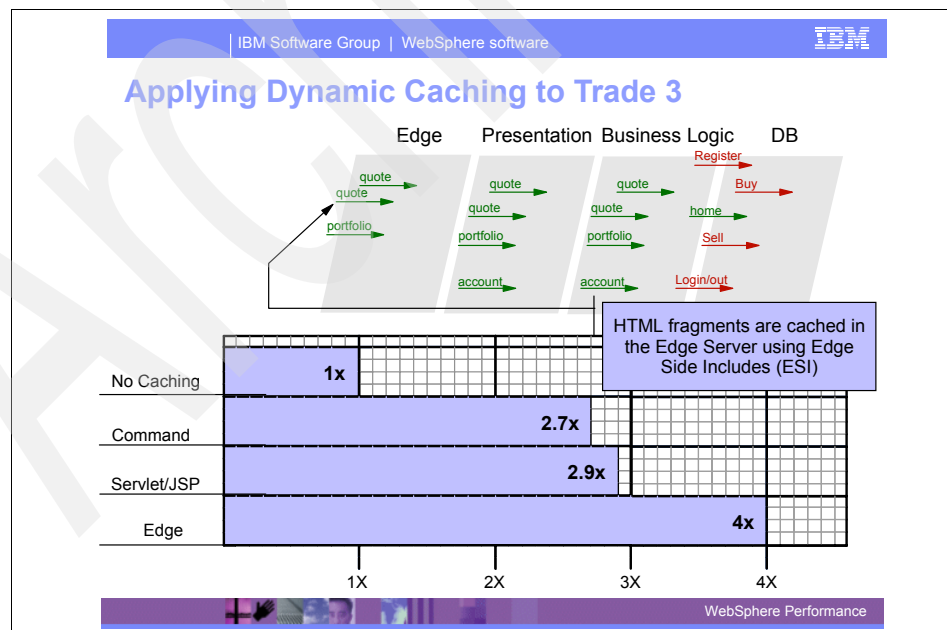


Figure 10-79 System performance with ESI processor caching

10.9.2 Edge Side Includes

Note: This section is extracted from the existing WebSphere Application Server Network Deployment 5.0 performance test. It discusses ESI performance considerations and a performance comparison for Trade3.

Performance

First, a threaded Web server is preferred over a strictly process-based Web server for two reasons:

1. Since there is a separate instance of the ESI processor's cache for each process, a threaded Web server with fewer processes allows a higher degree of cache sharing, and thus a higher cache hit ratio, lower memory consumption, and increased performance.
2. If `ESIInvalidationMonitor` is set to true (that is, if invalidations flow back to the ESI processor), then a long-running connection is maintained from each process to the `ESIInvalidatorServlet` in each application server cluster. Furthermore, each of these connections uses a thread in the application server to manage the connection. Therefore, a threaded Web server uses far fewer connections to the back-end application server, and far fewer threads to manage these connections.

Warning: It was noted from our testing of Trade3 on RedHat Linux Advanced Server 2.1 that more than 100 threads per process had adverse affects. In particular, the cost of using `pthread_mutex` locks (as is used by the ESI processor) with a large number of threads introduced a CPU bottleneck. By configuring IHS 2.0 to have 100 threads per process, `pthread_mutex` contention was minimized and the CPU utilization was maximized. The optimal thread-to-process ratio may differ for your application and will depend upon the cost of computing the data to be cached.

Figure 10-80 on page 615 shows a performance comparison of Trade3 on Windows with:

- ▶ No caching
- ▶ Servlet and command caching, but no ESI caching
- ▶ Servlet, command, and ESI caching

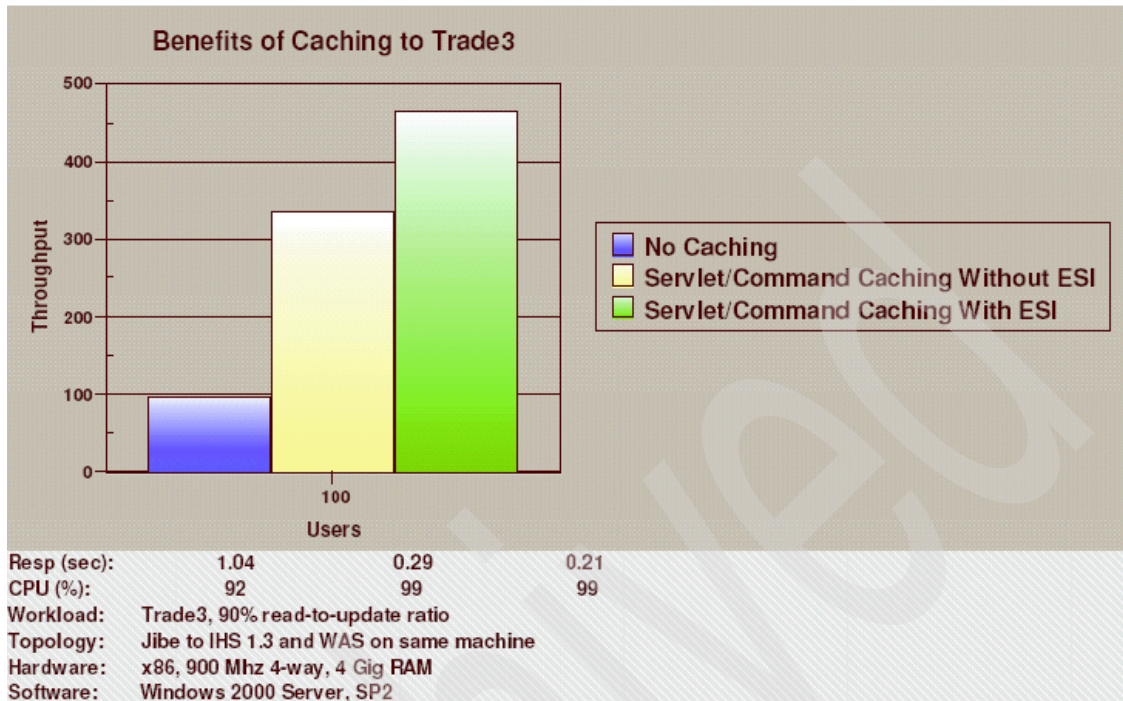


Figure 10-80 Benefits of caching

Figure 10-80 shows that servlet and command caching boost performance approximately 250%. ESI boosts the performance another 38% with a 90% read-to-update ratio for Trade3.

Best Practices

This section contains a set of best practices for making your application's fragments "edge cacheable" (that is, cacheable on the "edge" by the ESI processor in the WebSphere Web server plug-in).

- Modify the MVC (Model-View-Controller) design practices to isolate work into individual fragments. For example, the Trade3 home page provides personalized account information for an individual user as well as generic market summary data. To efficiently edge cache this output, the page is broken into two fragments. The personalized data is in tradeHome.jsp which is cached for each individual user, whereas the market summary page is cached as a single instance and shared by all users. To take advantage of edge caching, the work to produce the data for each page must be done in the page itself verses in a single controller servlet which does all of the work.

- ▶ Further fragment pages when necessary. For example, the Trade3 quotes page provides quotes for an arbitrary list of stocks entered by the user. The page provides an HTML table with each row containing the price and other information for an individual stock. To take advantage of edge caching, a single JSP fragment page is created for the rows in the Quote.jsp page. This new page, displayQuote.jsp, computes the current information for a given stock symbol. This allows all stocks to be individually cached at the edge. The ESI processor will assemble the page fragments corresponding to each individual stock row to create the full table.
- ▶ Unlike the dynamic cache service which runs within the WebSphere application server, the ESI processor does not have access to user HTTP session data to uniquely identify page fragments. The application must be designed such that page fragments can be uniquely identified using request parameters on the URL, HTTP form data or HTTP cookies in the request. In a JSP include, parameters should be included in the URL as query parameters instead of as JSP parameter tags, thus allowing the ESI processor visibility to these values as query parameters.
- ▶ Consideration should be given as to how expensive a given fragment is to compute. Fragments which are expensive to compute provide the best candidates for edge caching and can provide significant performance benefits. The cache entry sizes for Dynamic Caching and the ESI processor should be large enough to accommodate these expensive fragments. Also, the priority (or the time-to-live value) of these fragments should be raised to ensure less expensive fragments are removed from the cache first.
- ▶ Another important consideration for edge caching is the update rate of a page. Invalidation of cached fragments is a relatively expensive operation. Therefore, very dynamic fragments which are invalidated often may not benefit from caching, and may actually hurt performance. Most Web application pages, however, are quasi-static and thus can benefit greatly from Dynamic Caching and edge caching in WebSphere.

10.10 Reference

For additional information, see the following documents.

Note: Although these documents are based on WebSphere V5.x, they explain the dynamic caching concept very well and thus are also helpful for a WebSphere V6 environment.

- ▶ WebSphere Dynamic Cache: Improving J2EE application performance

<http://www.research.ibm.com/journal/sj/432/bakalova.pdf>

- ▶ IBM WebSphere Developer Technical Journal: Static and dynamic caching in WebSphere Application Server V5

http://www.ibm.com/developerworks/websphere/techjournal/0405_hines/0405_hines.html

- ▶ Exploiting Dynamic Caching in WebSphere Application Server 5.0, Part 1

<http://www.e-promag.com/eparchive//index.cfm?fuseaction=viewarticle&ContentID=3623&publicationid=19&PageView=Search&channel=2>

- ▶ Exploiting Dynamic Caching in WebSphere Application Server 5.0, Part 2

<http://www.e-promag.com/eparchive//index.cfm?fuseaction=viewarticle&ContentID=4409&publicationid=19&PageView=Search&channel=2>



Part 5

Messaging

Archived

Archived

Using asynchronous messaging for scalability and performance

The Java Messaging Service (JMS) has become very important in developing J2EE applications. It is used by disparate components within an IT environment to communicate both synchronously and asynchronously and also as a communication mechanism between two parts of the same J2EE application.

The specification for a J2EE 1.4 application server does not specify how messages should be represented or transported, but it does strictly specify the interface to the underlying messaging provider. JMS is the standard interface which a Java application uses to access a J2EE 1.4 compliant messaging provider. Every messaging provider in IBM WebSphere Application Server V6 - the default messaging provider, IBM WebSphere MQ, and the V5 default messaging provider - supports the JMS 1.1 API.

11.1 Introduction

This chapter covers the use of JMS from the perspective of scalability and performance. It will take the reader through an introduction to JMS 1.1, components and workflow of asynchronous messaging for large applications, and optimized JMS code development.

Although some fundamental concepts are reviewed, some sections of this chapter assume a previous understanding of JMS messaging.

11.2 Basic use of the JMS API

This section provides an introduction to using the JMS 1.1 API to access a J2EE 1.4 compliant message provider. It does not cover architectures for using JMS within applications. For an overview of the architecture of the Java Messaging Service, see:

<http://www.theserverside.com/articles/article.tss?l=JMSArchitecture>

11.2.1 The unified programming interface

JMS 1.1 provides a programming interface which combines the JMS 1.0.2b programming interface for point-to-point messaging (using queues) and publish-subscribe messaging (using topics):

Table 11-1 JMS 1.1 API

Unified or parent API	Point-to-point	Publish-subscribe
ConnectionFactory (CF)	QueueConnectionFactory (QCF)	TopicConnectionFactory (TCF)
Connection	QueueConnection	TopicConnection
Destination	Queue	Topic
Session	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueBrowser/QueueReceiver	TopicSubscriber

The unified API is recommended for new applications; however, JMS 1.1 supports the point-to-point and publish-subscribe API for backwards compatibility with legacy applications.

Example code

The code snippet in Example 11-1 steps through the use of the unified JMS API for message production.

Example 11-1 JMS 1.1 code snippet

```
[0]  try {
[1]
[2]      InitialContext context = new InitialContext();
[3]
[4]      ConnectionFactory connFactory = (ConnectionFactory)
[5]          context.lookup("java:comp/env/jms/MyConnectionFactory");
[6]
[7]      Connection conn = connFactory.createConnection();
[8]
[9]      boolean isTransacted = false;
[10]     Session sess = conn.createSession(isTransacted, Session.AUTO_ACKNOWLEDGE);
[11]
[12]     Destination des = (Destination) context.lookup("java:comp/env/jms/textQ");
[13]
[14]     MessageProducer producer = sess.createProducer(des);
[15]
[16]     TextMessage message = sess.createTextMessage("Hello World from WebSphere 6!");
[17]     producer.send(message);
[18]
[19] } catch(NamingException ne) {
[20]
[21]     logger.error("Naming error in MyServlet before sending Hello World", ne);
[22]
[23] } catch(JMSEException jmse) {
[24]
[25]     logger.error("Error associated with sending Hello World in MyServlet", jmse);
[26]     Exception linked = jmse.getLinkedException();
[27]
[28]     if(linked != null) {
[29]         logger.error("Error linked to JMS Exception in MyServlet", linked);
[30]     }
[31] }
```

Important: The try-catch block in this code sample does not close JMS resources. It is written under the assumption that the resources are still needed and does not include a finally block to close any unneeded resources. Refer to 11.4.4, “Freeing JMS object resources” on page 641 for more information.

A walkthrough of the code with references to the line numbers follows:

1. [Line 2] An initial context is created to look up references to JMS administered objects such as a connection factory or destination. The default context is used, which is the context of the environment the code is in during runtime, for example, the JNDI namespace tree for an application server.
2. [Lines 4-5] A known J2EE context reference (“java:comp/env/jms/MyConnectionFactory”) is used to retrieve a Java reference to an administered object. In this case, the administered object is the corresponding connection factory associated with the desired messaging provider. This context reference resolves to a JNDI name specified outside of the code. The JNDI name can be changed within the Administrative Console in the event that the administered object changes providing a level of indirection.

Note: Support for looking up resources directly by using the JNDI namespace is deprecated in WebSphere Application Server V6. This is where a lookup string without the “java:comp/env/” prefix is used and treated as a JNDI name.

3. [Line 7] The factory is used to create a JMS connection to the messaging provider.
4. [Lines 9-10] A session is created from the connection which is not transacted. Automatic acknowledgement of received messages is enabled (this argument is ignored if the session is transacted).
5. [Line 12] A J2EE context lookup is used to retrieve a Java reference to the textQ administered object which is casted as a `javax.jms.Destination` even though it is more specifically a `javax.jms.Queue` in the JMS provider. This allows for the same API to be used for both publish/subscribe and point-to-point.
6. [Line 14] A producer is created for the destination.
7. [Lines 16-17] A JMS text message is created and sent.

For a complete discussion of using the JMS 1.1 API, refer to section 10.2, “Java Message Service” of the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451).

A reference for the changes since JMS 1.0.2b is found in the developerWorks® article “JMS 1.1 simplifies messaging with unified domains,” found at:

<http://www.ibm.com/developerworks/java/library/j-jms11/>

11.2.2 Consuming JMS messages

When an application is written to access an asynchronous messaging system through JMS, it is typically in one of two ways:

- ▶ Generic JMS usage

Using the JMS API to send and receive messages. The application relies on WebSphere Application Server to locate the relevant messaging system and to handle the connections to it. What the application does with sending and receiving the messages is entirely up to the application developer within the bounds of the JMS API. For example, a `javax.jms.MessageConsumer` may be used to poll a JMS destination manually.

- ▶ JMS and message-driven beans (MDB)

A message-driven bean is an Enterprise JavaBean (EJB) which has the ability to consume asynchronous messages. It may consume a message received from a message listener or the JMS resource adapter. For a complete discussion of developing message-driven beans, see section 10.4, “Message-driven beans” of the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

Using the JMS API directly with a message consumer to poll destinations is not recommended over using message-driven beans. The introduction of message-driven beans in J2EE 1.3 provided a standard for application servers to reduce the complexity in consuming messaging associated with manually polling with a `MessageConsumer`.

Note: If you are familiar with the use of the `javax.jms.MessageListener` interface, the use of the `setMessageListener` and `getMessageListener` methods for `javax.jms.MessageConsumer` are no longer supported in J2EE Web or EJB containers. However, they are supported within Client containers.

11.3 Choosing what format to use within JMS messages

The choice of physical format within a JMS message is a key decision in architecting a messaging solution. In small messaging applications, using the various JMS message types (`BytesMessage`, `TextMessage`, `StreamMessage`, `MapMessage`, or `ObjectMessage`) can be sufficient. It is often the case in enterprise solutions for a `javax.jms.BytesMessage` or `javax.jms.TextMessage` to be used in addition to physical formatting in order to facilitate more complex data structures.

The effect of an efficient physical format is not vital for performance; however, small gains can be achieved with efficient and effective message representation. The subject is covered here for completeness.

Important: The size of a message body is also a performance consideration. A modest gain in performance may be gained from reducing the size of a heavily used JMS message in an application. In general, a series of small messages each with a specific purpose can perform better than a single verbose message. JMS 1.1 added `BytesMessage.getBodyLength()` which may prove useful in logging message payloads.

The following physical formats are the most common:

► XML

This is the most common message representation. It is an open standard which can be used by many modern platforms and enterprise systems. XML can be mapped to nearly any data representation using XSLT. It is popular for its ability to represent data in a format which both computer systems and people can read with standard tools.

However, there is a trade-off in additional payload associated with XML. It is text based with verbose, tagged formatting which is not an efficient physical format.

► Tagged delimited

A tagged delimited physical format uses a delimiter such as a colon to separate text fields. A tag, which is often a keyword, is used to identify the type of data a field contains. An example of a tagged delimited physical format is SWIFT which has been used in many large-scale enterprise applications.

► Record-oriented

A record-oriented physical format contains different types of fields with variable length. For example, a message may include an integer spanning two bytes and a string spanning 24 bytes. A struct in C is an example of a record-oriented physical format. A record-oriented message can provide the most efficient message, but comes at the cost of readability, maintainability, and inter-operability.

For a complete discussion of message formatting see the book *Enterprise Messaging Using JMS and IBM WebSphere* by Kareem Yusuf, Ph.D.. An excerpt on physical message formatting is available online at:

<http://www.phptr.com/articles/article.asp?p=170722>

11.4 Managing workload for asynchronous messaging

In 11.2, “Basic use of the JMS API” on page 622 we saw how the JMS components are used. This only showed a single request, where in reality there would be many simultaneous requests. A Java application server’s ability to handle multiple *threads* of work allows an application to perform well when the workload requires dealing with simultaneous requests. In a Web application, simultaneous user requests are serviced by Web container threads. If the Web application makes use of JMS, then JMS components corresponding to the local messaging provider will also be required to handle simultaneous requests.

For the receiving application using MDBs to consume the workload, it may also be optimal to have messages processed in parallel.

All the workload that arrives, either for MDBs or a MessageConsumer, needs to be efficiently handled within the bounds of the physical resources available to the application server and the underlying messaging system (and also any back end applications accessed through messaging).

There are administrative settings associated with each JMS messaging provider that allow controls to be placed on the workload, allowing a balance to be reached while providing the best throughput and response times for the physical hardware the application is running on.

11.4.1 Basic workload patterns

The way in which work arrives at the application server determines how many simultaneous JMS requests will be handled. Knowing the high level workflow of asynchronous messaging can be very helpful in administering applications with performance and scalability in mind.

Workload from Web or EJB container

When requests come from the Web or EJB container it is the number of threads in these containers that is controlling the upper limit for the number of simultaneous accesses to the JMS resources. This is assuming that the Web and EJB container thread pools have not been configured to allow growth beyond the maximum pool size specified.

Important: As the demand for artifacts in the Web and EJB container which produce JMS messages increases, the number of JMS connections and sessions needed increases.

When using the default messaging provider, the WebSphere MQ provider, or a generic provider which has maximum settings for JMS resource related pools, it is important to fine tune such settings for optimal performance.

Figure 11-1 and Figure 11-2 on page 629 show a typical way in which requests might flow from an application. They illustrate the flow for WebSphere MQ and the default messaging provider respectively. In this example all access to the connection factory is through the EJB container and messages are only placed on the destination. On each set of requests denoted by the arrows, is a tag that represents the controlling factor on how many simultaneous requests can reach that particular resource.

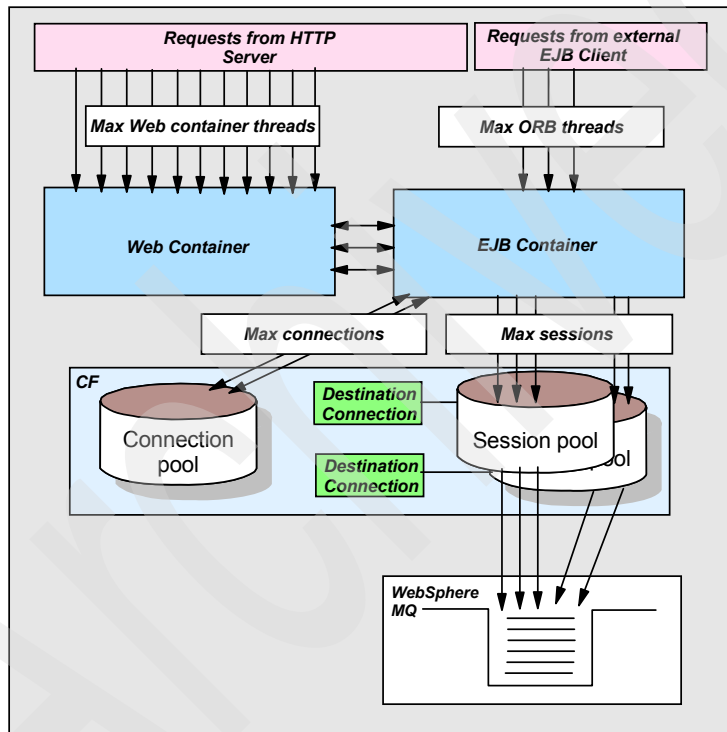


Figure 11-1 Request flow of EJB container accessing MQ JMS destinations

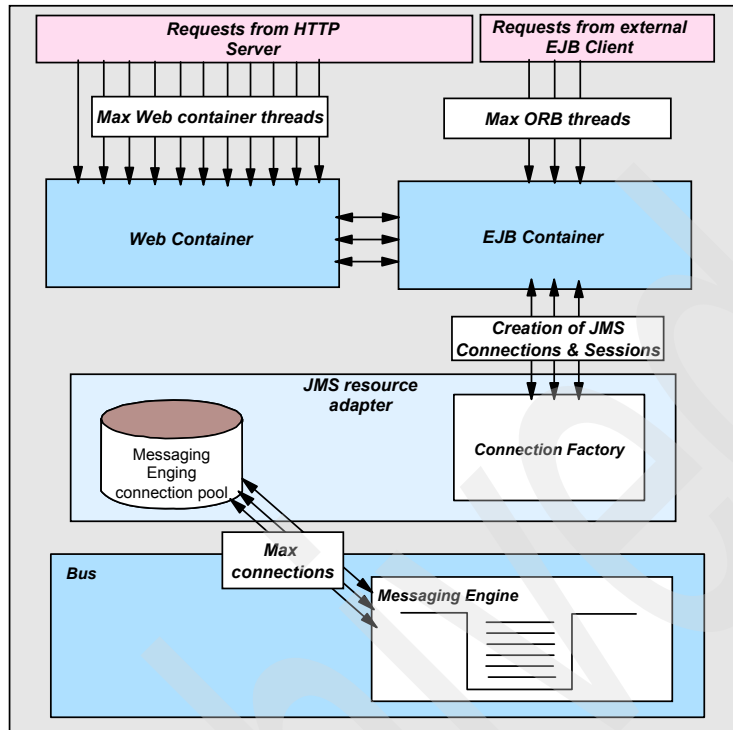


Figure 11-2 Request flow of EJB container accessing default provider destinations

As requests arrive, a proportion of them require use of JMS resources. There has to be enough of these resources available to handle all the requests. If there are not enough JMS resources available, then requests will have to wait for them to become free. This could lead to time-outs and re-submission of requests.

Note: If a JMS connection is not available and the JMS client is blocked, this blocks the corresponding EJB or Web container thread that is processing the request causing a subsequent lack of resources to process HTTP and EJB requests. This should be avoided at all costs by allocating enough resources on the messaging tier to handle the expected volume of request with an additional safety buffer.

Ideally, each request should only ever require the use of one JMS connection and one session at any time. Programming in this manner avoids deadlock situations. JMS 1.1 has made this even easier with the ability to access multiple types of destinations from the same session. As stated in the J2EE 1.4 specification, it is recommended that no more than one session be created per JMS connection.

Both the WebSphere MQ provider and the default messaging provider implement pooling mechanisms to facilitate management of JMS resources. The MQ JMS provider has maximum JMS session and connection pool settings in its connection factory settings. The Connection pool and Session pools for WebSphere MQ can be found in the WebSphere Administrative Console under **Resources -> JMS Providers -> WebSphere MQ -> WebSphere MQ connection factories**. The appropriate links are then found in the Additional Properties pane. See Figure 11-3 and Figure 11-4.

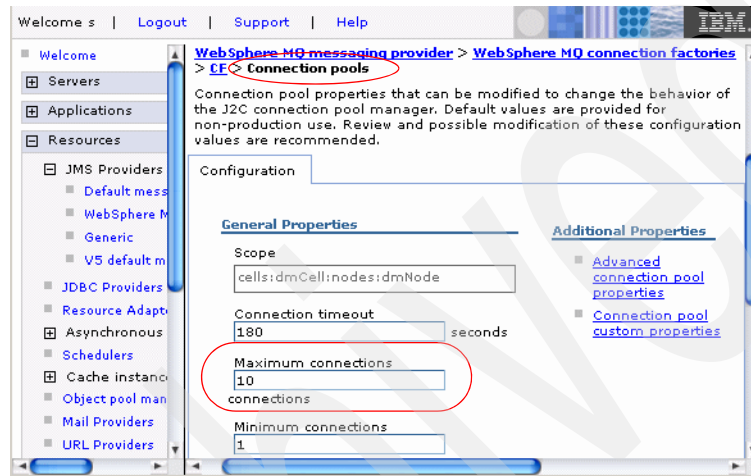


Figure 11-3 MQ JMS provider maximum connection pool setting

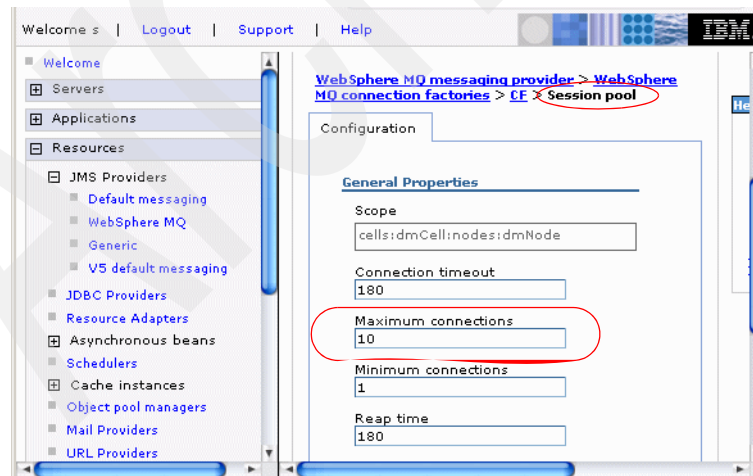


Figure 11-4 MQ JMS provider maximum connection pool setting

Important: Every MQ JMS connection has its own pool of session objects. This means that if the connection pool has a maximum of 5 and the session pool has a maximum of 10, providing the application is written to do this, there can be up to 50 Sessions open, 10 on each of the connections.

The pooling mechanism for the default messaging provider is different from that for the WebSphere MQ JMS implementation (or the V5 default provider implementation). For WebSphere MQ JMS, the application server provides a proxy to the JMS provider's objects enabling it to provide security and transactional integration. The application server also uses this additional layer of indirection to implement pooling of the provider's JMS connection and session objects.

In contrast, the default messaging provider is implemented as a J2EE Connector Architecture (JCA) resource adapter. The provider therefore uses the JCA interfaces to access the services provided by the application server. The application then accesses the provider's JMS objects directly and the provider pools a lower level connection to the messaging engine. This messaging engine connection is retrieved from the pool when a JMS connection is created and then associated with the first JMS session created from that connection. Subsequent JMS sessions will retrieve their own messaging engine connection from the pool. The messaging engine connection is returned to the pool when the JMS session, method, or current transaction (assuming a resource sharing scope of Shareable is configured) is closed.

The details of the underlying default messaging provider's JMS implementation are beyond the scope of this discussion - what is important for an administrator or architect is that balancing the maximum connection pool setting is important for performance and scalability. In WebSphere Application Server version 6.0.0.1 this setting can be found under **Resources -> Resource Adapters -> SIB JMS Resource Adapter -> J2C connection factories -> <connection_factory_name> -> Connection pool properties**. See Figure 11-5 on page 632.

Tip: Upcoming versions of WebSphere V6 might provide this setting under **Resources -> JMS Providers -> Default messaging -> JMS connection factory**.

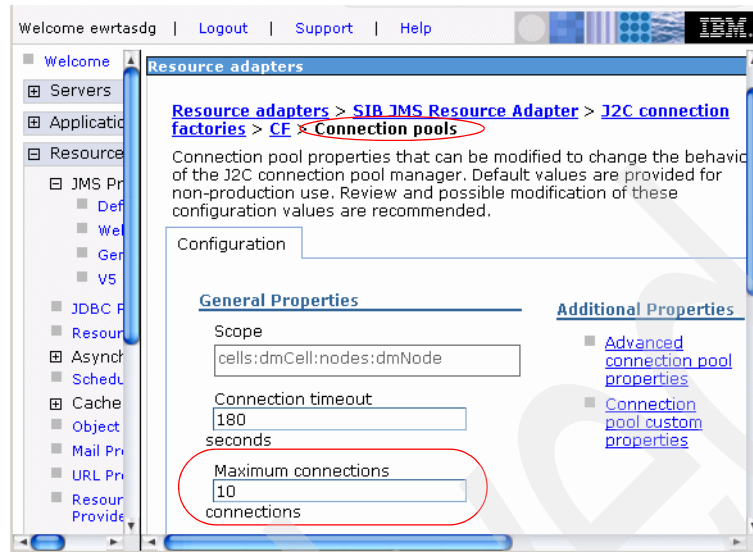


Figure 11-5 Default messaging provider maximum connection pool setting

More information about settings that can affect performance can be found in 12.6, “Choosing optimal configuration settings” on page 663 for the default messaging provider and 13.3, “Choosing optimal configuration settings” on page 708 for the WebSphere MQ provider. For information about monitoring these values in real time see 12.10, “Monitoring performance with Tivoli Performance Viewer” on page 693 for the default messaging provider and 13.6, “Monitoring performance with Tivoli Performance Viewer” on page 765 for the WebSphere MQ provider.

Workload from messaging system (MDB)

In this pattern, the workload is generated by messages to be consumed by message-driven beans. This means that the amount of workload that comes into the application server is directly associated with how many messages can be consumed by MDBs concurrently. Since the mechanism for consuming JMS messages using MDBs is different for the default messaging provider and the WebSphere MQ provider they are covered separately.

WebSphere MQ JMS provider

Figure 11-6 on page 633 illustrates the workflow of consuming messages with the MQ JMS provider. A message listener service monitors destinations and passes new messages to the corresponding message-driven bean. The relevant settings to be considered in tuning performance for message consumption are:

- Maximum sessions in session pool on connection factory

- ▶ Maximum connections in connection pool on connection factory
- ▶ Maximum sessions on listener port (JMS server sessions are part of the application server facilities, not JMS sessions)
- ▶ Maximum threads on message listener service
- ▶ Maximum EJB container pool size

Important: Tuning these settings is dependant on knowing the number of listener ports defined in an application server and the optimal maximum for sessions on each listener port.

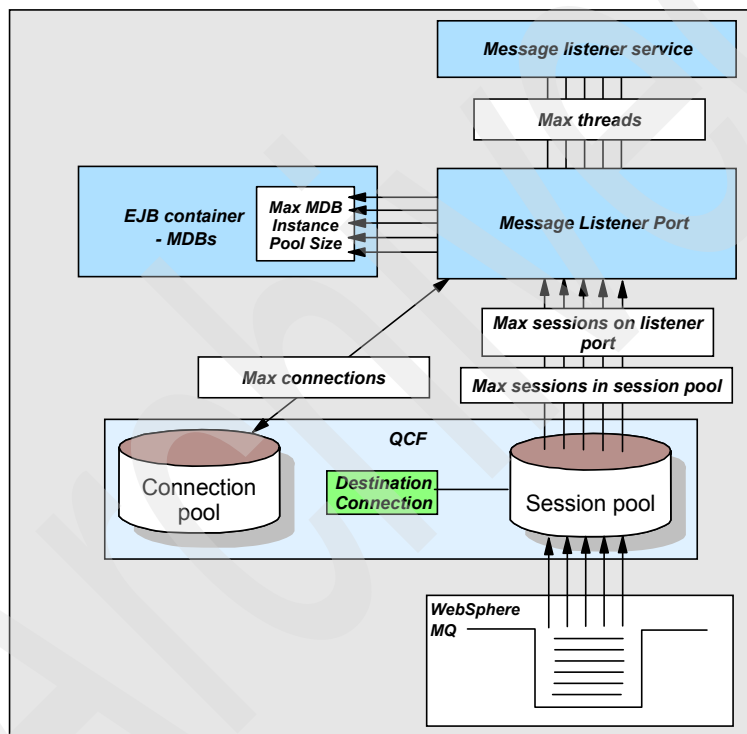


Figure 11-6 Request flow when using MDBs for the MQ JMS provider

If the maximum sessions for a listener port is set to 5, then there can be up to five sessions working in parallel to process messages on the message destination. The default for this value is 1 which would in effect make messages be processed in a serial fashion. Setting it to any more than 1 means that the order that messages get processed in cannot be guaranteed.

Explanation: Three listener port sessions are working off one message destination with three messages on it, each session picks up a message. There is a chance that session 2 could finish first, before sessions 1 and 3 have completed. This would mean message 2 is processed before message 1.

Transaction rollback would also cause messages to be processed in a different order. Same scenario, but this time message 2 is rolled back. It is put back on the destination, and so could end up being processed after message 3 has completed. If you have maximum sessions set to 1, then message 3 won't be processed until message 2 has been successfully processed.

As this setting governs how much simultaneous activity will occur in the application server it also governs how much of the other resources will be needed. When a listener port session is in use it needs:

- ▶ One message listener service thread
- ▶ One Connection (a listener port uses one Connection regardless of the number of listener port sessions)
- ▶ One Session

Choosing the correct number of listener port sessions to handle the messages as they arrive defines all the other parameters.

Ideally, the number of sessions on a listener port must be sufficient to handle the peak load of messages arriving on the message destination. This would have to be balanced against the system resources and all the other processing that is going on in the application server.

The minimum and maximum amount of MDB instances in the EJB container that

Important: A listener port will use one Connection object regardless of the number of listener port sessions, and that object remains in use as long as the listener port is started.

A listener port session, once established will hold on to all the resources it needs until it is finished. This means it will be using up a Session and a message listener service thread.

run at a single point in time (referred to in Figure 11-6 on page 633) can be configured with the `com.ibm.websphere.ejbcontainer.poolSize` system

property. For more information, search for “EJB container system properties” in the WebSphere Application Server V6 Information Center:

<http://www.ibm.com/software/webservers/appserv/infocenter.html>

For more information about settings related to the WebSphere MQ provider, such as configuring connection factories, see 13.3, “Choosing optimal configuration settings” on page 708.

Information about monitoring the WebSphere MQ provider can be found in 13.6, “Monitoring performance with Tivoli Performance Viewer” on page 765.

Default messaging provider

Figure 11-7 on page 636 illustrates the workflow of consuming messages with the default messaging provider. A JMS resource adapter monitors destinations and passes new messages to the corresponding message-driven bean. The relevant settings to be considered in tuning performance for message consumption are:

- ▶ Maximum batch size for the JMS activation specification
- ▶ Maximum concurrent endpoints for the JMS activation specification
- ▶ Maximum EJB container pool size

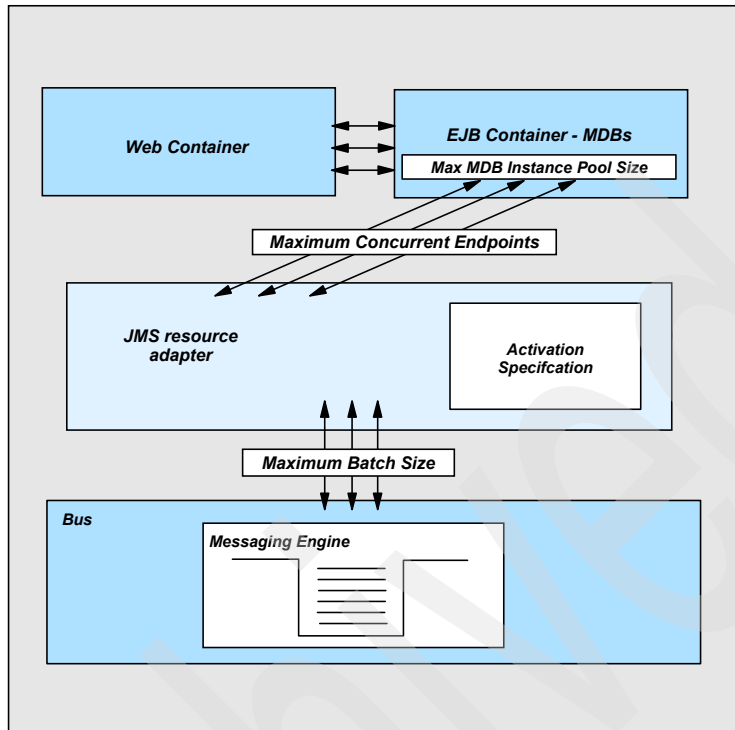


Figure 11-7 Request flow when using MDBs for the default messaging provider

Note: The activation specification does not pass messages to MDBs or poll destinations. It is a collection of settings for the JMS resource adapter exposed in the Administrative Console which configure the adapter to perform asynchronous message delivery to a message endpoint.

If this picture looks unfamiliar, a full discussion of these components can be found in sections 10.3 and 10.4 of the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

The workflow is somewhat simplified for the default messaging provider compared to the MQ JMS provider. However, there are still settings which control how much concurrent activity is allowed when messages are consumed.

The batch size configures the maximum number of messages in a single batch delivered serially to a single message-driven bean instance. Batching of messages can improve performance particularly when used with Acknowledge

mode set to Duplicates-ok auto-acknowledge. If message ordering must be retained across failed deliveries, set the batch size to 1 (default).

Increasing the number of concurrent endpoints can improve performance but can increase the number of threads that are in use at any one time. If message ordering must be retained across failed deliveries, set the maximum concurrent endpoints to 1 (default is 10).

The minimum and maximum amount of MDB instances in the EJB container that run at a single point in time (referred to in Figure 11-7 on page 636) can be configured with the `com.ibm.websphere.ejbcontainer.poolSize` system property. For more information, search for “EJB container system properties” in the WebSphere Application Server V6 Information Center:

<http://www.ibm.com/software/webservers/appserv/infocenter.html>

Ideally, the number of concurrent endpoints and the EJB container pool must be sufficient to handle the peak load of messages arriving on the message destination. This needs to be balanced against the system resources and all the other processing that is going on in the application server. Information about how to monitor these settings is found in 12.10, “Monitoring performance with Tivoli Performance Viewer” on page 693.

More information about workload considerations for the default messaging provider is covered in 12.5, “Clustering, high availability and workload management” on page 656.

More information about configuring the default messaging provider for performance can be found in 12.6, “Choosing optimal configuration settings” on page 663.

11.4.2 Selectors

A selector acts as a filter when waiting to receive a message. If a selector has been specified on the receive action then only messages that match criteria in that selector will be returned. A selector can be setup on MDBs and also coded in generic JMS when doing a receive.

Selectors in JMS point-to-point messages

The WebSphere MQ queue manager does not support filtering by content. So to implement JMS selectors in point-to-point, MQ JMS will browse a message, parse it, compare it with the selector and then either retrieve it if there is a match or browse the next message. This is termed *client-side selection*, and has a performance cost as each message must be retrieved to the client to be browsed.

Important: This is not the case for the default messaging provider. The default provider can access properties in an encapsulated message because its messages are formatted using the Service Data Object (SDO) framework. Thus, selectors on user specified properties do not require parsing the entire message payload.

WebSphere MQ does allow filtering by certain fields with certain formats in the MQ header which accompanies all messages. Two such fields are available to the JMS API:

- ▶ Message ID
- ▶ Correlation ID

When selecting on these fields, in the particular case of equality testing, the MQ JMS code is able to avoid the browse-parse-get loop and simply do a get-by-messageId or get-by-correlationId. This optimization is termed *server-side selection* as the messages are no longer retrieved to the client first to attempt to match the selector, instead this is done on the server. This only applies if the selector is a simple one, so for example no use of AND or OR with another field in the selector string. Using the server-side selection makes the matching process run faster.

The server side selection will only work if the values for message ID or correlation ID match the messaging provider's JMS format.

Message ID

A message ID is generated by the JMS provider and has to conform to the JMS specification of:

ID:nnnnn

where nnnnn is a provider specific value. For WebSphere MQ the message ID is a 48 character hexadecimal value and will look like:

ID:414d51205741535f617070315f6a6d73b9f1cc3f2000071c

As it is generated, writing a selector that uses the message ID field will use server-side selection.

Correlation ID

The correlation ID is not generated by the JMS provider and is left to the programmer to set. To take advantage of server-side selection the correlation ID needs to be set to the correct format so that WebSphere MQ will recognize it and bypass the costly browse-parse-get loop. There are two ways to do this:

- ▶ Set the correlation ID to the message ID

The message ID is already in the correct format as it is generated. Using this saves having to construct a correlation ID of the correct format and is the recommended method.

- Manually construct the correlation ID to the correct format

This means making a String that is 48 hexadecimal characters. If you wish to use a String that is readable and then convert that into hexadecimal then that String needs to be 24 characters long.

Example

One common activity which uses a selector is within one application thread, send a request message and wait for the reply message. The reply message is identifiable on the incoming destination by an identifying field, in this example the correlation ID field. Finding the correct message is then a case of using a selector that will match on that correlation ID.

The code to do this would look something like the code fragment in Example 11-2.

Example 11-2 Example receive using a selector string for correlation ID

```
Session sess = conn.createSession(false, Session.AUTO_ACKNOWLEDGE);
String messageSelector = "JMSCorrelationID = '"+correlid+"'";
MessageConsumer consumer;
consumer= sess.createConsumer(myDestination, messageSelector);
TextMessage message = (TextMessage) consumer.receive();
String s = message.getText();
```

This code will only do a server-side selection if the field `correlid` is encoded in the correct format. The most common approach is for the receiving application to copy the message ID of the request into the correlation ID. This can be done with the `setJMSCorrelationID` method for a `javax.jms.Message`.

MDBs and selectors

The selector for an MDB is set in the EJB deployment descriptor. Unlike the previous discussion of generic JMS and selectors, the selection process is done by the application server and messaging provider not by the code written by the J2EE application developer. The message is automatically delivered when a match according to the selector is found.

Keeping the depth of the queue (number of messages on a queue) to a minimum always increases the speed of using message selectors as there are less messages which participate in the selection mechanism. When using MDBs with message selectors in point-to-point, you should always make sure to have all messages that arrive on a destination dealt with.

Any messages that do not match the selection criteria for the MDBs corresponding to that destination, will be left sitting on the queue. Over time they could build up and slow down the selection process. To avoid this, either make sure that the MDBs that use selectors on a message destination, cover all possibilities for arriving messages, or use one of the configuration options to remove the unselected messages.

Depending on your application, it might be possible to specify an expiration for messages on the destination.

If WebSphere MQ is the provider, an alternative to handling all cases in the application code is to disable message retention. The message listener service will check all the selector strings of its MDBs against each message that arrives. If it does not find a match then the default is for this message to be returned to the destination. This should not be used for applications that cannot tolerate message loss of any kind.

This setting is configurable in the Administrative Console for a connection factory under the WebSphere MQ JMS provider.

To disable message retention, uncheck the box next to **Enable message retention** then save the configuration. See Figure 11-8.

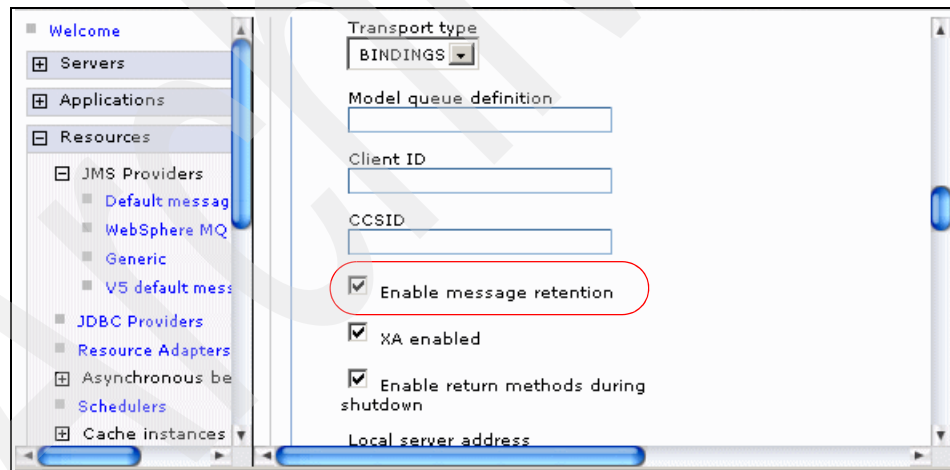


Figure 11-8 Disable message retention on an MQ connection factory

Important: This setting changes the behavior for all parts of the application that are accessing a destination on the WebSphere MQ provider through this connection factory. Use this setting only when you are confident that it will not affect any other part of the application. Also, each destination can only have one process that is using selectors on it through this connection factory. If there is more than one process, for example the message listener service and a MessageConsumer running in a Web container thread, then there is a danger that messages will be lost.

11.4.3 Application defined persistence

Important: It is not recommended that you make decisions regarding persistence at the application code level. Persistence can be configured in the Administrative Console independent of the code facilitating maintainability and reuseability. The subject is covered here only for completeness.

Using persistence on a message destination should only be used when needed if performance is a requirement. WebSphere MQ and the default messaging provider have the option to leave destination persistence decisions at the application level by setting the persistence for a queue or topic as APPLICATION DEFINED.

The `javax.jms.MessageProducer` interface includes `setDeliveryMode()` which can be used to set a persistent or non-persistent delivery mode.

Important: By default, persistence is turned on for a message producer.

11.4.4 Freeing JMS object resources

When using JDBC to access a database it is a best practice to make sure that all objects are closed after their use - even when exceptions occurs. The same applies when using JMS. Figure 11-3 shows an example of how the finally block after a try-catch should be coded after a JMS connection is no longer needed.

The close method for `javax.jms.Connection` triggers a close for all JMS objects below it in the hierarchy. For example, closing a `QueueConnection` (a subinterface of `Connection`) will close a `QueueSender` or `QueueSession` created under that connection. Closing the highest object in the JMS hierarchy which is no longer needed is recommended.

Example 11-3 Sample finally block for a piece of code that sends a message

```
finally {  
    // Close JMS Connection  
    try {  
        if (connection != null) {  
            connection.close();  
            connection = null;  
        }  
    } catch (JMSEException jmse) {  
        logger.error("Connection close() failed within MyServlet", jmse);  
        if(jmse.getLinkedException() != null)  
            logger.error("linked cause for error", jmse.getLinkedException());  
    }  
}
```

Using and optimizing the default messaging provider

In WebSphere Application Server V6, the embedded messaging server from WebSphere Application Server V5.x was replaced by the default messaging provider, which is supported by the Service Integration Bus, a new component of WebSphere Application Server V6.

Throughput testing of the default messaging provider has yielded unmatched results compared to WebSphere MQ and the V5.x embedded messaging server in a subset of possible configurations. If maximum throughput is desired and low message reliability is acceptable, using the default messaging provider may be the optimal choice for performance.

This chapter describes how the various components of the default messaging provider interact, including bus members and messaging engines. It takes you through manually configuring the necessary JMS components for the Trade 6 application and demonstrate running Trade 6 on the default messaging provider architecture.

For information about IBM WebSphere MQ as a messaging provider for WebSphere Application Server V6, see Chapter 13, “Understanding and optimizing the use of WebSphere MQ” on page 697.

12.1 Introduction

Performance and stability of an application using JMS are largely governed by:

- ▶ Efficient and safe application usage of JMS
- ▶ The most efficient messaging and application server topology, but also one that provides adequate failure handling
- ▶ Optimal configuration settings of each individual application server and its resources

Stability is also important for maintaining performance; the application will need to be able to cope with failures in a graceful manner. If it is unable to do this then performance will degrade and the service provided by the application becomes unacceptable.

It is the aim of this chapter to provide the reader with enough information to cover these areas, either through discussion or by identifying other documents that already cover a particular area.

This chapter is structured so that each section builds on the knowledge from previous sections. It starts with taking a look at what happens when you use the default messaging provider in your application. This is so that the following sections can use this to help build the larger picture of the implications of making configuration changes within a default messaging provider setup.

While some terminology and concepts are described within this chapter in completeness, others might require a previous understanding of the technology behind the default messaging provider. If you are new to JMS, then you should first look at chapter 10, “Asynchronous messaging”, of the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

Also, the Trade 6 application is used in our examples, so familiarity with this application allows you to better understand what is being described. Some information about how Trade 6 uses JMS can be found in 12.8.1, “What does Trade 6 use the default messaging provider for?” on page 673. Also, refer to the document *tradeTech.pdf* coming with the Trade 6 download package. Trade 6 can be downloaded from:

<http://www.ibm.com/software/webservers/appserv/performance.html>

Note: At the time of writing this redbook, Trade 6 was not yet available for download. It is expected soon. Please monitor this page for availability.

12.2 Introduction to the Service Integration Bus and the default messaging provider

This section gives you a brief introduction on the concepts and architecture of the Service Integration Bus and the default messaging provider in WebSphere Application Server V6. For detailed information, please refer to Chapters 10, “Asynchronous messaging” and 11, “Default messaging provider” in the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

The Service Integration Bus provides a managed communications framework which supports a variety of message distribution models, reliability options and network topologies. It provides support for both message-based applications and service oriented architectures.

The Service Integration Bus acts as the default messaging provider for WebSphere Application Server V6.

By using a messaging bus, which is a cell wide resource rather than a single messaging hub like the WebSphere Application Server v5.x embedded JMS Server, clients are isolated from the actual topology/implementation details to the underlying messaging provider. For example, a client application running on a server which is a bus member, does not need to know a host name to connect to the messaging provider or any particular destination. Thus, the topology of the bus can be changed by an administrator without changing the clients.

12.2.1 Bus or Service Integration Bus

A bus or Service Integration Bus is a component of WebSphere Application Server V6 that supports applications using message-based and service-oriented architectures.

Note: Hereafter, the Service Integration Bus is referred to as the bus or messaging bus.

A WebSphere cell can contain any number of buses. To be of any use, each bus should contain one or more bus members. The exact decision as to how many buses and bus members you have depends on your messaging needs and cell topology. Some systems may require only a single bus with a single bus member, where other systems may require or benefit from a more complex messaging topology.

Benefits of multiple messaging engine busses include:

- ▶ Improved scalability by spreading messaging workload across multiple servers.
- ▶ Performance enhancements from situating messaging destinations close to message producing or consuming applications.
- ▶ Improved availability by removing single points of failure.

The ability to have multiple buses within a single cell can be beneficial. It enables completely separate messaging systems to be used for different purposes, such as having test and production systems where destination names need to be the same, or the separation of messaging applications which have no need to communicate with each other.

12.2.2 Bus members and messaging engines

When you add an application server to a bus, a messaging engine is created on the server. A messaging engine is part of the server that manages messages and messaging connections for a particular bus. A server might be a member of more than one bus, in which case the server will have one messaging engine for each bus that it is a member of.

When clients connect to a bus, they will be connected to it through a messaging engine.

When you add a server cluster to a bus, a messaging engine will also be defined, however, unlike a J2EE application, the messaging engine will not be active on all servers within the cluster at the same time, it will only be active on one of them. If, for any reason, the messaging engine, or the server it is running on, should fail, the messaging engine will be restarted on another available server in the cluster. See 12.5, “Clustering, high availability and workload management” on page 656 for more details.

Messaging engines are created for you by the bus and have generated names. These names follow a strict format, as follows:

- ▶ For a bus member that is a server, the messaging engine name has the format `<nodeName>.<serverName>-<busName>`.
- ▶ For a bus member that is a cluster, the messaging engines names have the format `<clusterName>.<XXX>-<busName>`. Where `XXX` is a number from 000 upwards. This value will be the lowest number available that does not conflict with existing numbers.

12.2.3 Destinations

Bus destinations are logical addresses within a bus to which applications can attach as message producers or message consumers.

Note: Although loosely related, a bus destination is not a `javax.jms.Destination`, the JMS interface which JMS queue and topic objects implement.

There are four different types of destinations:

- Queue

A queue is a destination which is configured for point-to-point messaging. While a queue is a bus wide resource, a queue will have a *queue point*, created when the queue is defined, which is associated with a single bus member. This bus member is responsible for hardening (writing to disk) any messages that need to be persisted. It is beneficial, for performance reasons, to have message consumers such as message-driven beans running in the same application server as the queue point for a queue being consumed from.

- Topic space destinations for publish/subscribe messaging

A topic space is a destination which can be used for publish and subscribe messaging. A topic space is also a bus wide resource but, unlike a queue, has no association with any particular bus member. Within a topic space, clients can publish to any named topic, for example `news/sports/golf`. Clients can subscribe to named topics, or use a wildcard to select a group of topics such as `news/sports/*`.

- Alias destinations act as an alias to another destination

For example, a queue named *a* may be accessed by using an alias destination named *b*. These also provide a mechanism for overriding default values, such as default reliability and security, for clients accessing a particular destination through the alias.

- Foreign destinations

Foreign destinations are a way of overriding foreign bus default values for particular destinations on a foreign bus.

12.2.4 JMS activation specification

When a message-driven bean (MDB) is deployed as a Java Connector Architecture (JCA) 1.5 resource as a listener on the default JMS messaging provider, it needs a JMS activation specification providing the necessary

configuration information to receive messages. The configuration includes the name of the bus and the JNDI name of the JMS destination.

One or more message-driven beans can be associated with a JMS activation specification.

For more information about how to configure a JMS activation specification, please refer to the WebSphere Application Server V6 InfoCenter at:

<http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp>

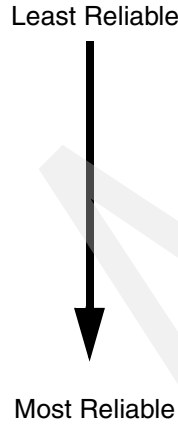
Refer to the Java Connector Architecture (JCA) 1.5 documentation for more information about activation specification and resource adapters at

<http://java.sun.com/j2ee/connector/download.html>

12.2.5 Message reliability

Messages on the bus have a property called reliability which defines how the messages will be delivered. There are five levels of reliability, as detailed in Table 12-1.

Table 12-1 Message reliability



Reliability	
Best Effort nonpersistent	Messages are discarded when a messaging engine is stopped, or if it fails. Messages may also be discarded if a connection used to send them becomes unavailable or as a result of constrained system resources. Messages delivered asynchronously to non-transactional MessageListeners or message-driven beans will not be redelivered if an exception is thrown.
Express nonpersistent	Messages are discarded when a messaging engine is stopped or if it fails. Messages may also be discarded if a connection used to send them becomes unavailable.
Reliable nonpersistent	Messages are discarded when a messaging engine is stopped or if it fails.
Reliable persistent	Messages may be discarded if a messaging engine fails, but not when it is stopped.
Assured persistent	Messages are never discarded.

Reliability should be chosen according to your messaging needs, more reliable messages will perform less well than less reliable messages.

The exact reliability of any particular message is configured in a number of ways:

1. The delivery mode of the JMS MessageProducer can be set programatically by the JMS client to `PERSISTENT` or `NON_PERSISTENT`.

This will be overridden by:

2. The optional `deliveryMode` parameter on the `JMS MessageProducer.send()` method.

This will be overridden by:

3. The default messaging provider JMS destination resource which can specify whether the delivery mode can be specified by the JMS application or whether the persistence will be set to `persistent` or `nonpersistent`.

This will be overridden by:

4. The bus destination which can be configured to disallow message producers to override the default message reliability that is set on the bus destination. The default is for a bus destination to allow message producers to override the default reliability.

The default messaging provider JMS connection factory used by the JMS application to connect to the bus provides the mapping of the `PERSISTENT/NON_PERSISTENT` delivery mode option in JMS into the various levels of reliability provided by the bus. The Quality of Service properties on the connection factory allow you to specify the desired mapping of JMS `PERSISTENT` messages and JMS `NON_PERSISTENT` messages to the reliability provided by the bus. It is also possible to specify that the default reliability of the bus destination should be used by selecting “As bus destination”.

12.2.6 Data stores

Every messaging engine has a data store. A data store is a set of database tables that are exclusively accessed by a messaging engine and used to harden (write to disk) messages that need to be persisted. They are accessed using a JDBC data source, the JNDI name of which must be configured in the messaging engine.

A messaging engine prevents other messaging engines from accessing the same data by storing a unique identifier (UUID) in the database.

12.3 Components used in a default messaging provider configuration

This section helps you to understand exactly what components are involved when using the default messaging provider to access a messaging system.

When an application is written to access an asynchronous messaging system through JMS, it is typically in one of two ways:

- ▶ Using the JMS API to send and receive messages

The application relies on WebSphere Application Server to locate the relevant messaging system and to handle the connections to it. What the application does with sending and receiving the messages is entirely down to the application developer within the bounds of the JMS API.

- ▶ Message-driven beans (MDB)

The application relies on WebSphere Application Server to monitor the underlying messaging system and pass any new messages to an instance of a purpose built MDB that will handle the message.

12.3.1 JMS component diagram for sending a message

Putting a message onto a message destination using JMS requires a number of components. Figure 12-1 on page 651 depicts how the ServletToDestination servlet places a message on a topic or queue. The code is written to work with either the point-to-point or publish/subscribe model using the unified JMS API. For an introduction to the unified JMS 1.1 API see 11.2, “Basic use of the JMS API” on page 622 which provides a complementary reference.

For this example, the following are needed:

- ▶ JNDI Naming service

Described in detail in chapter 13 of the *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451 redbook.

- ▶ Connection Factory (CF)

Encapsulates the settings necessary to connect to a messaging system.

- ▶ Destination

A reference to the destination.

- ▶ A message queue or topic

The actual destination where messages are held in the messaging engine until removed by an application.

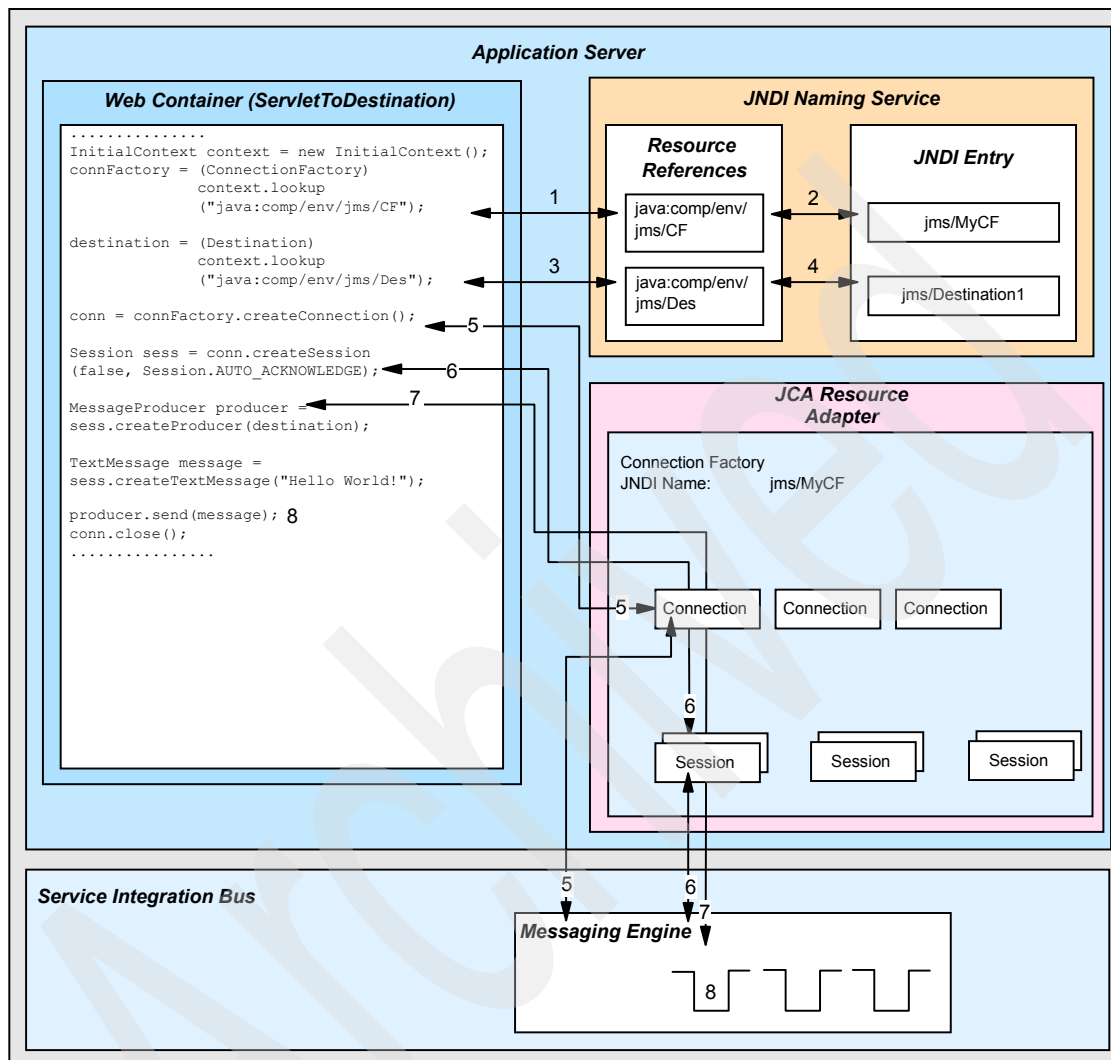


Figure 12-1 Component interactions to send a message using JMS 1.1 API

1. The servlet is invoked by the user and the `doGet` method is called by the Web container.
2. The application needs to locate the correct destination to place the message on. The JNDI namespace is used to house a link to the Java objects that will perform communication with the messaging system. The `ServletToDestination` servlet uses *resource references* within the code. These are linked at application deployment time to the JNDI entries that can be used to communicate with the messaging system.

3. The code performs a JNDI lookup (1) for the connection factory using the resource reference.
4. The resource reference is matched to the JNDI entry that contains the connection factory(2). The connection factory (java:comp/env/jms/CF) object is returned, it will be used to get a connection to the messaging engine that is responsible for the target destination.
5. A second JNDI lookup is performed (3) and resolved to the correct JNDI entry (4). This time it is for the destination, this will be used to locate the required destination from the messaging engine.
6. To be able to communicate with the queue in the messaging system, the application must first create a connection (5) from the ConnectionFactory object. This request to create a new JMS connection will result in the JCA resource adapter creating (or obtaining from the pool) a managed connection to the messaging engine.
7. A session object is created from the connection (6). The managed connection from the connection is then associated with the session. When the session is closed the managed connection is returned to the pool. See “Workload from Web or EJB container” on page 627 for more information about pooling.
8. With a connection now established to the messaging system through the session object, the application now specifies what sort of action is going to be performed, in this case it is to send a message (7). The destination definition that was taken from the JNDI name service is passed to the session object and this gives the messaging engine information for the specific destination on to which the message will be placed.
9. The message is constructed and sent to the message destination(8).

This is just a basic example of how JMS might be used within an application. However, it demonstrates the number of components that are used to send a message with the default messaging provider. The configuration of these components can be crucial in making sure that the application performs fast enough for requirements.

12.3.2 Bus and message-driven beans

Using message-driven beans with the default messaging provider incorporates further components because the application server is now responsible for delivering the message to the application.

Message-driven beans (MDBs) attached to destinations in the bus are attached by means of the bus JCA resource adapter for JMS, an activation specification which defines the settings for the adapter, and a JMS destination. The resource

adapter is responsible for connecting to the bus and delivering messages to the MDB.

As an example, the Servlet2MDBQueue servlet, which can be found in the Trade 6 application, places a message on a queue associated with the JNDI name `jms/TradeBrokerQueue`. Within the default configuration that comes with Trade 6, an activation specification “`eis/TradeBrokerMDB`” has been defined that configures the bus resource adapter to monitor the queue. Any messages that arrive on the queue are passed to the TradeBrokerMDB to be processed. This process is shown in Figure 12-2 on page 654.

For this example, the following are used:

- ▶ The components listed in “JMS component diagram for sending a message” on page 650:
 - JNDI name service
 - Connection factory (CF)
 - Destination
 - A message queue
- ▶ Message-driven bean (MDB)

When a message arrives on a queue that is being monitored by a corresponding resource adapter, the `onMessage` method of its associated MDB is called and the MDB will consume that message.

- ▶ Activation specification

Each MDB using the default messaging provider is associated with an activation specification which contains the configuration for the JMS resource adapter to invoke the MDB when the message arrives.

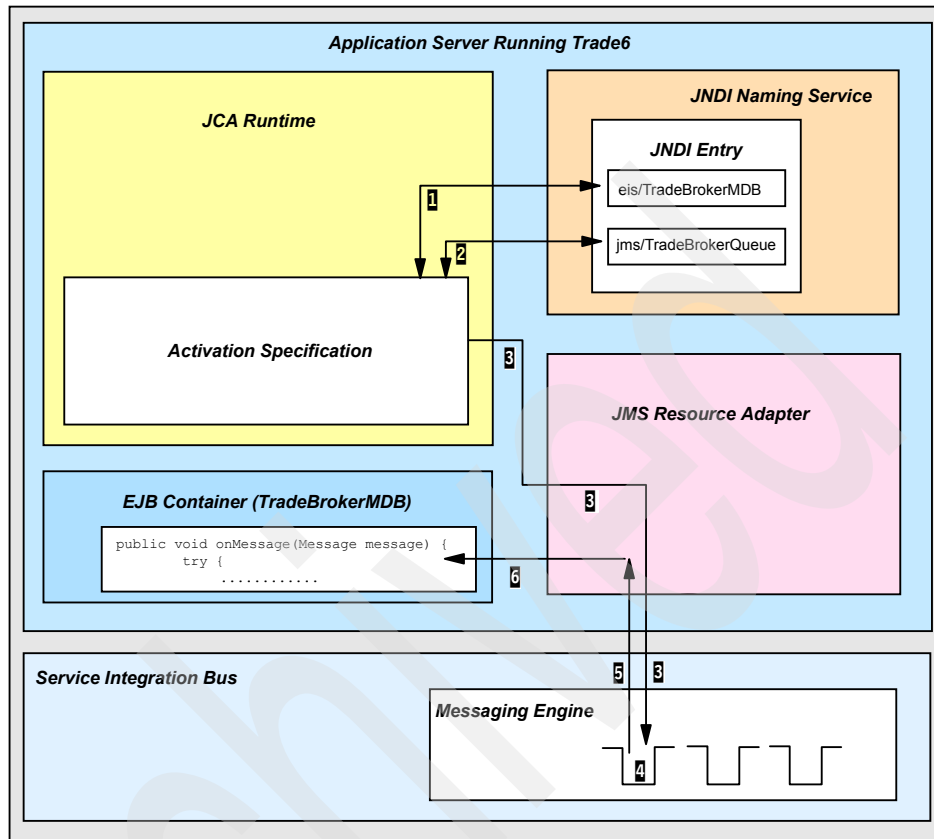


Figure 12-2 Component interactions receiving a message from a queue using a MDB

1. The MDB is associated with the activation specification that is in turn associated with the correct queue. Upon initialization the JCA runtime performs a JNDI lookup (1) for the activation specification. The activation specification is returned, it will be used to get a connection to the messaging engine that is responsible for the target queue.
2. The JCA runtime also performs a JNDI lookup for the queue destination (2) that the activation specification has been configured with. This will be used to locate the required queue from the messaging engine.
3. The JCA runtime provides the activation specification to the JMS resource adapter which then connects to the messaging engine (3). Any messages arriving on the message queue will be picked up one or more at a time (depending on the batch size) and processed by the correct MDB.
4. A message arrives (4) on the queue.

5. As the resource adapter is ready to accept messages, it will take delivery of the message off the queue (5). The resource adapter then passes the message on to an instance of the MDB that it is associated with. The `onMessage` method is called on the MDB passing it the message.

12.4 Component relationships

Compared to WebSphere MQ, the component relationships in the default messaging provider are greatly simplified (see 13.2, “MQ JMS component relationships” on page 705). The JMS resource adapter has replaced all components related to the message listener service. Pooling related to JMS connections and sessions has been consolidated to one JCA connection pool which pools connections to the messaging engine. Figure 12-3 illustrates the component relationships for the default messaging provider.

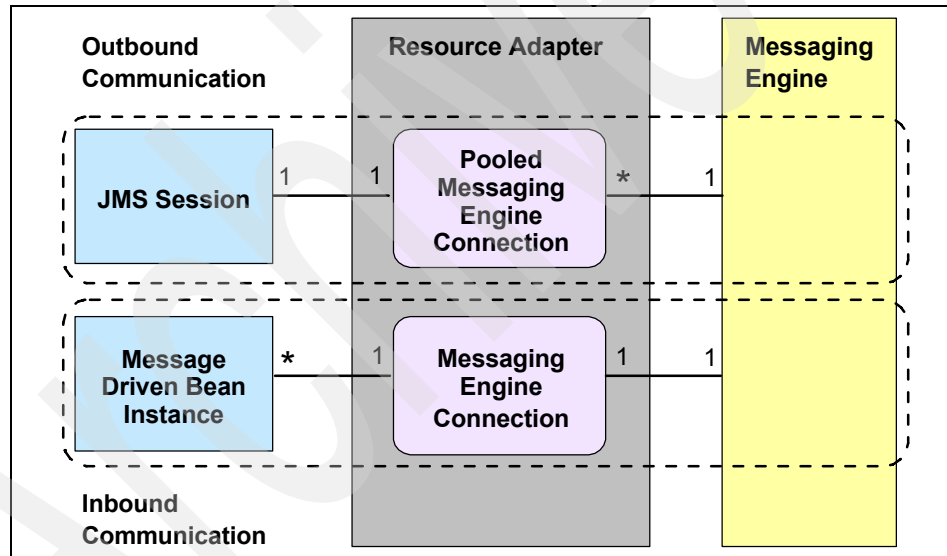


Figure 12-3 Component relationship diagram

For Web and EJB container workloads, high volume JMS use means using more JMS connections and sessions, whether it is workload using the publish and subscribe or point-to-point model. Refer to 11.4.1, “Basic workload patterns” on page 627 for more information. This requires more JCA connections to the messaging engine in the resource adapter. Also, heavy workflow inbound to an application can strain the amount of concurrent JMS endpoints corresponding to the consuming message-driven bean.

These relationships should be considered when tuning an application for performance. 12.10, “Monitoring performance with Tivoli Performance Viewer” on page 693 should be referred to when tuning for performance.

12.5 Clustering, high availability and workload management

The bus provides high availability and workload management for messaging. It is worth noting however, that bus messaging engines do not follow the same clustering model that J2EE applications do in WebSphere Application Server clusters.

A J2EE application (such as an EJB or servlet) that is installed on a cluster of application servers runs on all active application servers in the cluster simultaneously. Thus, workload management is provided for the application. If one of the servers fails then the application will still be available on the other servers which provides high availability for the application.

12.5.1 Cluster bus members for high availability

When an application server cluster is added to a bus as a cluster bus member, the messaging engine defined will be highly available. The messaging engine becomes active on only one server within the cluster. Should the messaging engine fail, or the server it is running on fail or be stopped, then the messaging engine will automatically be started on another server in the cluster if one is available.

Since the messaging engine is only available on one server within the cluster, there is no workload management of the messaging function provided by the cluster bus member.

By default, the messaging engine will start on the first available server in a cluster. If you wish to ensure that the messaging engine runs on one particular server in the cluster, for example if you have one primary server and one backup server, or if you wish the messaging engine to only run on a small group of servers within the cluster, then you must specifically configure this. See 12.5.5, “Preferred servers and core group policies” on page 660 for details on configuring preferred servers.

12.5.2 Cluster bus members for workload management

To achieve greater throughput of messages, it is beneficial to spread the messaging load across multiple servers, and optionally across multiple hosts. You can achieve this, while maintaining a simple destination model, with a cluster of messaging engines, each of which has a preference to run on a separate server in the cluster.

Once a server cluster has been added to a bus, that cluster is a cluster bus member. As mentioned earlier, a cluster bus member automatically gets one messaging engine defined on it, but it may optionally have additional messaging engines added to it. These messaging engines can be configured to prefer to run on separate servers within the cluster. This enables a messaging engine to run in every server in the cluster, providing every application in the cluster with a messaging engine for local access to the bus. Local access to the bus is always better for messaging performance, especially in the case of queues where the queue is assigned to the bus member being accessed.

When a queue is assigned to a cluster bus member, the queue will be partitioned across all messaging engines in the cluster.

12.5.3 Partitioned queues

A queue is partitioned automatically for you when a queue destination is assigned to a cluster bus member.

Every messaging engine within the cluster owns a partition of that queue and will be responsible for managing messages assigned to the partition. Every message sent to the queue will be assigned to exactly one of the partitions.

Local partitions

When a JMS client that is attempting to access a partitioned queue is connected to a messaging engine that hosts one of those partitions (a messaging engine in the cluster) then the client will only be able to access that “local” partition of the queue for both consuming and producing messages.

Note: The only instance where messages will not be sent to the local partition is when that local partition is full and other partitions of the queue are not. In this case, messages will be routed to a non-full remote partition.

Clients will only attempt to consume from the local partition, even if there are no messages in the local partition and there are messages available on other partitions.

Remote partitions

If the JMS client is connected to a messaging engine that does not host a partition of the destination (a messaging engine in the same bus that is not in the cluster), then each session that client creates will connect to one “remote” partition for the purposes of consuming messages. Each session created will be workload-managed with respect to which remote partition it connects for consuming messages.

Messages sent to a remote partitioned destination will be workload-managed across the individual partitions on a per-message basis regardless of the session.

Important: It is worth noting that cluster bus members and partitioned queues alone will not give better throughput of messages. The applications that are producing and consuming the messages must be configured to make use of the support provided by the bus.

► **Message producers** must be configured to ensure that messages they produce will be workload-managed onto different partitions of a partitioned queue. There are several ways of doing this, including:

- Message producers (JMS clients) connecting directly into the cluster, though this has some restrictions in version 6.0, see “JMS clients connecting into a cluster of messaging engines” on page 659.
- Message producers connecting to messaging engines that are not part of the cluster. This requires servers outside of the cluster to be available and added to the bus.
- Message production by an EJB/servlet installed on the cluster. Workload management of the calls to the EJB/servlet will workload manage the message production across the cluster as the EJB/servlet will connect to the messaging engine running in the same server. Messages produced by the EJB/servlet will then be routed to the local partition of the queue.

► **Message consumers** must be configured to connect to each partition of a partitioned queue to consume messages. If any partitions do not have consumers then the messages sent to that partition may never be consumed.

The simplest, and recommended way of configuring consumers to every partition of a partitioned queue is by installing a message-driven bean on the cluster.

12.5.4 JMS clients connecting into a cluster of messaging engines

JMS clients outside of a cluster can connect directly into a workload-managed cluster of messaging engines under the following three conditions:

1. The cluster is a bus member.
2. One messaging engine is added for each server in the cluster.
3. The policy to tie each messaging engine to a server is one-to-one.

The workload management is highly dependent on the application code. For example, if the application creates one `javax.jms.Connection` and uses it throughout its life span, the first messaging engine it connects to always receives the messaging load for the application server it is running on (assuming it is the only messaging application on the server). On the other hand, if the application creates a new JMS connection for each of its threads or for each incoming HTTP request, then the workload is balanced at a lower level and is far more dynamic.

JMS clients connecting in this way still use the connection rules described in section 10.7, “Connecting to a service integration bus” of the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451, however there is an undesirable side effect of these rules when the servers in the cluster are used as the provider endpoints for the connection factory. Consider the following example.

A JMS client connects into a cluster of servers A, B and C. The connection factory is configured with provider endpoints of A, B, C. This allows the client to bootstrap to any of the three servers in the cluster. Following the connection rules, the connection factory will bootstrap to the first server in the provider endpoints list, A. Server A has a local messaging engine, so the messaging engine on Server A will be chosen as the preferred connection point for the client.

Since the connection always tries the first entry in the provider endpoints list first, every client connecting directly into the cluster is thus connected to the messaging engine in server A. This is naturally not very good for workload management of messaging. There are two methods that can be used to overcome this:

- Enable a bus service located on a server outside of the cluster. Configure the provider endpoints to point to this bus service. If there is no messaging engine local to this bus service then the client connections will be workload-managed around all of the messaging engines in the bus.

If you only have messaging engines in the cluster no further configuration is required. If there are other non-cluster bus members and you only wish the clients to connect directly to the messaging engines in the cluster then you must also configure a target group on your connection factory. See “Target

groups” in section 10.7.2 of the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451 for more information.

- ▶ Provide different clients with differently configured connection factories, each of which has a different provider endpoint in the first position on the list.

12.5.5 Preferred servers and core group policies

To configure a messaging engine to prefer a server or group of servers you must configure a core group policy. A core group policy is used to identify server components, and define how they will behave within a cell or cluster. For more information about core groups and core group policies, please refer to Chapter 9, “WebSphere HAManager” on page 465 and chapter 11 “Default messaging provider” of *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

Policy type

For bus messaging engines, a policy type of *One of N* should be used. This means that while the messaging engine may be defined on every server in the cluster, WebSphere’s HAManager ensures that it is only active on one of the servers in the group, and will always be active on one of the servers as long as one is available.

Match Criteria

The match criteria of a core group policy enables the HAManager to decide what server components match the policy and therefore should be managed according to the policy. There are two match criteria that are used to match a messaging engine:

- ▶ `type=WSAF_SIB`. This matches any messaging engine.
- ▶ `WSAF_SIB_MESSAGING_ENGINE=<messaging_engine_name>`. This matches the messaging engine of the name provided.

Note: In order to override the default policy, `type=WSAF_SIB`, at least two match criteria are needed.

Preferred servers

The preferred servers defined in a policy allow you to list a group of servers that the messaging engine will prefer to run on. The higher up in the list of preferred servers a particular server is, the more preferred it is. For a messaging engine that is part of a cluster bus member you should only select preferred servers that

are actually part of the cluster. The messaging engines are only defined in the cluster and cannot be run on any servers outside of the cluster.

Fail back and Preferred servers only

These two options have a considerable effect on how a particular policy will make a messaging engine behave in a cluster.

Fail back

If *Fail back* is selected, when a more preferred server becomes available then the messaging engine will be deactivated where it is currently running and activated on the more preferred server.

Enabling Fail back ensures that a messaging engine always runs on the most preferred server that is available. This is usually desirable since there should be a good reason for configuring a preferred server in the first place.

If you do not enable Fail back, then once a messaging engine has started, it will not move to a more preferred server should one become available.

Preferred servers only

If *Preferred servers only* is selected then the messaging engine is only allowed to be active on servers in the policy's preferred servers list. If Preferred servers only is not selected, all servers in the cluster that are not in the list will also be able to have the messaging engine active on it, but they will be selected only if none of the preferred servers are available.

You should be very careful when selecting Preferred servers only because it is possible to reduce or remove the high availability of a messaging engine and of the partitions of queues that the messaging engine owns. If none of the preferred servers is available then the messaging engine will not be active anywhere which means that the partitions of any queues owned by that messaging engine will also be unavailable.

Any messages currently on those partitions will be unavailable and cannot be consumed until one of the preferred servers has become available and the messaging engine has been activated.

Large clusters

If you have a medium or large cluster of servers (for example five or more) configured with messaging engines then we recommend a special configuration of preferred servers.

With a large number of messaging engines defined on a cluster, it would be undesirable to have all of the messaging engines starting up on the first server in the cluster to be started, and so we suggest the following configuration:

- Each messaging engine should be configured with Fail back and Preferred servers only enabled.
- A group of preferred servers should be configured that is a sub-set of all of the servers in the cluster.

This subset must be large enough to provide sufficient failover capabilities for the messaging engine, at least two or three servers. Each messaging engine should be configured with a different sub-set of servers, with each messaging engine having a unique most preferred server. An example is shown in Figure 12-4.

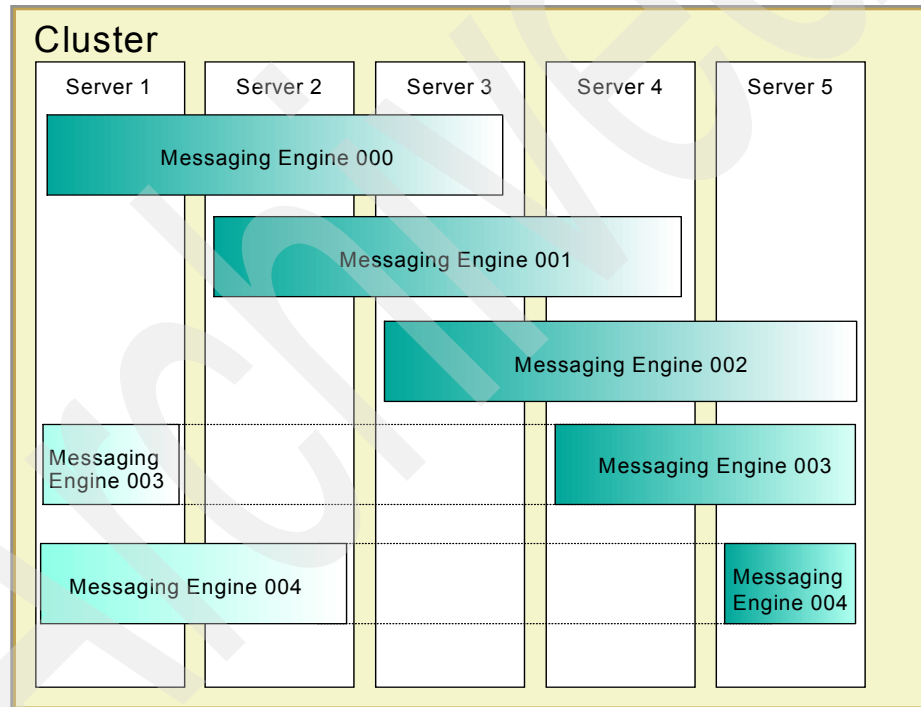


Figure 12-4 Configuring large clusters of messaging engines

12.6 Choosing optimal configuration settings

This section is a list of guidelines and settings that can help in finding the all important starting point for performance tuning an application. It is intended to help add some reason in choosing specific settings.

Important: Each application will operate differently in the way it uses the JMS components. Any values mentioned in this section should not be treated as a generic optimal value.

The configuration settings discussed in this section are:

- ▶ Important connection factory settings
- ▶ Setting up default messaging provider connection pools
- ▶ Important JMS activation specification settings

12.6.1 Important connection factory settings

There are several settings for the connection factory that could affect the performance:

- ▶ Target and Target type
- ▶ Target significance
- ▶ Connection proximity
- ▶ Non persistent and persistent message reliability
- ▶ Read ahead
- ▶ Share data source with CMP

Target and Target type

You can specify that the connection factory uses a subset of messaging engines on the bus with the `target` setting. The type of the target is specified in `Target type` and can be one of the followings:

- ▶ Bus member name
- ▶ Custom messaging engine group name
- ▶ Messaging engine name

If these settings are left unspecified, the default behavior is to connect to a local messaging engine whenever possible and to load balance connections to foreign messaging engines if no local engine is available. This is the recommended configuration. Thus, if you need the connection factory to use a specific member of the bus, you may want to override the default behavior by specifying these settings.

Target significance

This property specifies the significance of the target group with the following two choices:

- ▶ Preferred

It is preferred that a messaging engine be selected from the target group. A messaging engine in the target group is selected if one is available. If a messaging engine is not available in the target group, a messaging engine outside the target group is selected if available in the same bus.

- ▶ Required

It is required that a messaging engine be selected from the target group. A messaging engine in the target group is selected if one is available. If a messaging engine is not available in the target group, the connection process fails.

Connection proximity

Only messaging engines that are within the selected proximity setting can be chosen for applications to connect to.

For an application running in an application server, this property defines how close messaging engines must be to the application server. For an application running outside an application server, this property defines how close messaging engines must be to the bootstrap server.

When searching for the most suitable messaging engine a closer messaging engine is always selected ahead of a more remote messaging engine.

- ▶ Bus

Connections can be made to messaging engines in the same bus.

A suitable messaging engine in the same server is selected ahead of a suitable messaging engine in the same host, and in turn ahead of a suitable messaging engine in another host.

- ▶ Cluster

Connections can be made to messaging engines in the same server cluster. If the application is not running in a clustered server, or the bootstrap server is not in a cluster, then there are no suitable messaging engines.

A suitable messaging engine in the same server is selected ahead of a suitable messaging engine in the same host, and in turn ahead of a suitable messaging engine in another host.

- ▶ Host

Connections can be made to messaging engines in the same host. A suitable messaging engine in the same server is selected ahead of a suitable messaging engine in the same host.

- ▶ Server

Connections can be made to messaging engines in the same application server.

Non persistent and persistent message reliability

The reliability for both non persistent and persistent messages can be specified as follows:

- ▶ None

Use the delivery option configured for the bus destination.

- ▶ Best effort non persistent

Messages are discarded when a messaging engine stops or fails. Messages may also be discarded if a connection used to send them becomes unavailable as a result of constrained system resources.

- ▶ Express non persistent

Messages are discarded when a messaging engine stops or fails. Messages may also be discarded if a connection used to send them becomes unavailable.

- ▶ Reliable non persistent

Messages are discarded when a messaging engine stops or fails.

- ▶ Reliable persistent

Messages may be discarded when a messaging engine fails.

- ▶ Assured persistent

Messages are not discarded.

- ▶ As bus destination

Use the delivery option configured for the bus destination.

As mentioned in 12.2.5, “Message reliability” on page 648, the higher the reliability level you choose, the lower the performance will be.

Read ahead

Read ahead is an optimization technique used by the default messaging provider to reduce the time taken to satisfy requests from message consumers. It works by preemptively assigning messages to consumers.

Messages that are assigned to a consumer are locked on the server and sent to a proxy destination on the client, prior to the message consumer requesting them, the locked messages cannot be consumed by any other consumers for that destination.

Messages that are assigned to a consumer, but not consumed before that consumer is closed, are subsequently unlocked on the server and then available for receipt by other consumers.

By default this is enabled only for non-durable topic consumers and durable subscribers in a non-clustered environment. You can override this property for individual JMS destinations by setting the Read ahead property on the JMS destination. An example of when you may want to consider overriding the settings is when you know there is only a single consumer for a queue.

Share data source with CMP

Messaging engines store persistent data in a database, using a JDBC data source to interact with that database. Some JMS applications also store persistent data in a database, for example if the application uses entity Enterprise JavaBeans (EJBs). Typically, such applications use two-phase commit transactions to coordinate updates to the JMS and JDBC resources involved.

Share data source with CMP allows sharing of connections between JMS and Container Managed Persistence (CMP) entity EJBs. This has been estimated as a potential performance improvement of 15% for overall messaging throughput, but can only be used for entity beans running in the application server that contains the messaging engine.

Using this setting, the applications can be configured to share the JDBC connection used by a messaging engine, which enables the applications to use one-phase commit transactions and improve the performance of the applications.

You can benefit from the one-phase commit optimization in the following circumstances:

- ▶ Your application must use the assured persistent reliability attribute for its JMS messages.
- ▶ Your application must use CMP entity beans that are bound to the same JDBC data source that the messaging engine uses for its data store.

You cannot benefit from the one-phase commit optimization in the following circumstances:

- ▶ If your application uses a reliability attribute other than assured persistent for its JMS messages.

- If your application uses BMP entity beans or JDBC clients.

For a more complete discussion of sharing database connections between messaging and CMP see the WebSphere InfoCenter article “Sharing connections to benefit from one-phase commit optimization”.

Administering durable subscriptions

Durable subscriptions created by message-driven beans are not removed automatically by the application server - the decision to remove unneeded subscriptions is done manually via the Administrative Console. This is covered in the WebSphere InfoCenter. Search for “Administering durable subscriptions”.

12.6.2 Setting up default messaging provider connection pools

The connection factory for the default messaging provider is managed by the bus JMS resource adapter. The pool settings for the messaging engine connections will affect the performance for JMS clients. You can configure these settings:

- Connection timeout
- Maximum connections
- Minimum connections
- Reap time
- Unused timeout
- Aged timeout
- Purge policy

To configure the connection pools settings, click **Resources** -> **Resource Adapters** in the console navigation tree, then click **SIB JMS Resource Adapter** -> **J2C connection factories** -> **<connection_factory_name>**. The pool settings are found under the Connection pool properties link.

Note: These are not JMS connections (`javax.jms.Connection`), but rather connections between JMS clients and the messaging engine.

Connection timeout

This value indicates the number of seconds a request for a connection waits when there are no connections available in the free pool and no new connections can be created before a `ConnectionWaitTimeoutException` is thrown, usually because the maximum value of connections in the particular connection pool has been reached.

Note: `ConnectionWaitTimeoutException` is not the exception the application will get, but is usually a linked exception to the `JMSEException`.

For example, if Connection timeout is set to 300, and the maximum number of connections are all in use, the pool manager waits for 300 seconds for a physical connection to become available. If a physical connection is not available within this time, the pool manager initiates a `ConnectionWaitTimeout` exception. It usually does not make sense to retry the `getConnection()` method; if a longer wait time is required you should increase the Connection timeout setting value. If a `ConnectionWaitTimeout` exception is caught by the application, the administrator should review the expected connection pool usage of the application and tune the connection pool and database accordingly.

If the Connection timeout is set to 0, the pool manager waits as long as necessary until a connection becomes available. This happens when the application completes a transaction and returns a connection to the pool, or when the number of connections falls below the value of Maximum connections, allowing a new physical connection to be created.

If Maximum connections is set to 0, which enables an infinite number of physical connections, then the Connection timeout value is ignored.

Maximum connections

This value specifies the maximum number of physical connections that you can create in this pool.

These are the physical connections to the message destination in the default messaging provider. Once this number is reached, no new physical connections are created and the requester waits until a physical connection that is currently in use returns to the pool, or a `ConnectionWaitTimeout` exception is issued.

For example, if the Maximum connections value is set to 5, and there are five physical connections in use, the pool manager waits for the amount of time specified in the Connection timeout field for a physical connection to become free.

If Maximum connections is set to 0, the connection pool is allowed to grow infinitely. This also has the side effect of causing the Connection timeout value to be ignored.

You can use the Tivoli Performance Viewer to find the optimal number of connections in a pool. If the number of concurrent waiters is greater than 0, but the CPU load is not close to 100%, consider increasing the connection pool size. If the `PercentUsed` value is consistently low under normal workload, consider decreasing the number of connections in the pool.

Minimum connections

This value specifies the minimum number of physical connections to maintain.

If the size of the connection pool is at or below the minimum connection pool size, the Unused Timeout thread does not discard physical connections. However, the pool does not create connections solely to ensure that the minimum connection pool size is maintained. Also, if you set a value for Aged Timeout, connections with an expired age are discarded, regardless of the minimum pool size setting.

For example if the Minimum connections value is set to 3, and one physical connection is created, the Unused timeout thread does not discard that connection. By the same token, the thread does not automatically create two additional physical connections to reach the Minimum connections setting.

Reap time

This value specifies the interval, in seconds, between runs of the pool maintenance thread.

For example, if Reap time is set to 60, the pool maintenance thread runs every 60 seconds. The Reap time interval affects the accuracy of the Unused timeout and Aged timeout settings. The smaller the interval, the greater the accuracy. If the pool maintenance thread is enabled, set the Reap time value less than the values of Unused timeout and Aged timeout. When the pool maintenance thread runs, it discards any connections remaining unused for longer than the time value specified in Unused timeout, until it reaches the number of connections specified in Minimum connections. The pool maintenance thread also discards any connections that remain active longer than the time value specified in Aged timeout.

The Reap time interval also affects performance. Smaller intervals mean that the pool maintenance thread runs more often and degrades performance.

To disable the pool maintenance thread set Reap time to 0, or set both Unused timeout and Aged timeout to 0. However, the recommended way to disable the pool maintenance thread is to set Reap time to 0, in which case Unused timeout and Aged timeout are ignored. However, if Unused timeout and Aged timeout are set to 0, the pool maintenance thread runs, but only physical connections which timeout due to non-zero timeout values are discarded.

Unused timeout

This value specifies the interval in seconds after which an unused or idle connection is discarded. This can only happen to connections in the pool that are free or not currently associated with a JMS resource.

You should set the Unused timeout value higher than the Reap timeout value for optimal performance. Unused physical connections are only discarded if the current number of connections exceeds the Minimum connections setting. For

example, if the Unused timeout value is set to 120, and the pool maintenance thread is enabled (Reap time is not 0), any physical connection that remains unused for two minutes is discarded. Note that accuracy of this timeout, as well as performance, is affected by the Reap time value. See Reap time for more information.

Aged timeout

This value specifies the interval in seconds before a physical connection is discarded. This is the total age of the connection from when it was created, not just the time since it was last used.

Setting Aged timeout to 0 supports active physical connections remaining in the pool indefinitely. Set the Aged timeout value higher than the Reap timeout value for optimal performance. For example, if the Aged timeout value is set to 1200, and the Reap time value is not 0, any physical connection that remains in existence for 1200 seconds (20 minutes) is discarded from the pool. Note that accuracy of this timeout, as well as performance, are affected by the Reap time value. See Reap time for more information.

Purge policy

This setting specifies how to purge connections when a stale connection or fatal connection error is detected. Valid values are EntirePool and FailingConnectionOnly.

12.6.3 Important JMS activation specification settings

There are several JMS activation specification settings that can affect the performance.

Maximum batch size

The maximum number of messages received from the messaging engine in a single batch.

The maximum number of messages in a single batch delivered serially to a single message-driven bean instance. Batching of messages can improve performance particularly when used with Acknowledge mode set to Duplicates-ok auto-acknowledge. If message ordering must be retained across failed deliveries, set the batch size to 1.

Maximum concurrent endpoints

The maximum number of endpoints to which messages are delivered concurrently.

Increasing this number can improve performance but can also increase the number of threads that are in use at any one time. If message ordering must be retained across failed deliveries, set the maximum concurrent endpoints to 1.

12.7 Failure to process a message

The JMS resource adapter retrieves the messages from the message queue and passes them to the MDB. If there is an error in processing a message then the MDB will rollback and the JMS resource adapter returns that message to the queue. Of course, once back on the queue this message will be picked up again by the JMS resource adapter and a never ending cycle could occur.

There are two settings that can stop this occurring:

- Maximum failed deliveries

On each messaging destination (topic or queue) it is possible to configure the *maximum failed deliveries* setting. The default value is 5.

- Exception destination

When the maximum failed deliveries is reached, the message will be forwarded to the destination configured as the *Exception destination*. By default, the failed message is forwarded to the system exception destination of the messaging engine that discovered the problem:

`_SYSTEM.Exception.Destination.<engine_name>` but you can also specify another queue on the same bus or a foreign bus.

To configure these two settings select **Service integration -> Buses -> <bus_name> -> Destinations -> <queue_name>. See Figure 12-5.**



Maximum failed deliveries
5

Exception destination

System ☒

None ☐

Specify ☐ \$DEFAULT_EXCEPTIC

Figure 12-5 Maximum failed deliveries and Exception destination settings for a messaging destination

It is possible to change the maximum retries to a higher number to give the message a chance to succeed. For example, if the message was being used to update a database but access to the database timed out then retrying, more retries might eventually work.

12.8 Usage scenarios: Trade 6 and the default messaging provider

A lot of the settings we discussed have topology specific benefits. Functionality and configuration are a lot more significant when designing a messaging and application server topology for more than traditional Web and EJB based applications. With those it is the performance and scalability requirements that often force the choice of topology.

When using JMS, there is not one topology at the top of the food chain that will suit all situations. Using point-to-point, publish/subscribe, MDBs, different messaging products, and combinations of these will all drive a need to have a specific topology to match a specific application and its non-functional requirements.

The choice of topology will depend on the correct balance of non-functional requirements of performance, security, scalability, maintenance, reliability, availability and cost, as well as providing a topology that can give the required function.

The example application that has been used throughout this book is Trade 6. Trade 6 has specific requirements for its use of JMS. Trade 6 uses JMS as an asynchronous communication mechanism between two parts of the application. This means that for Trade 6 there is no need to communicate with a back-end system and so a lot of the more complicated topologies do not suit it.

However, this chapter is supposed to be helping you understand JMS, so for the purposes of this section, two scenarios are described using Trade 6 with slightly different requirements for functionality and other criteria. The front end sections of the topology (Web traffic) are not discussed but it is possible to overlay the sample topology in Chapter 8, "Implementing the sample topology" on page 387.

This section is intended to help you use some of the configuration options described in this chapter in working examples. It should also help you begin to understand the thought process in deciding on the best topology for your application. It will not describe all possible topologies, and there are many other topologies. To find out more on choosing a topology take a look at the document *JMS Topologies and Configurations with WebSphere Application Server and WebSphere Studio Version 5* available at:

http://www.ibm.com/developerworks/websphere/library/techarticles/0310_barci a/barcia.html

Note: This document is currently being updated for WebSphere V6 and will become available on developerWorks in the near future.

12.8.1 What does Trade 6 use the default messaging provider for?

Before continuing to the examples, it is important that the functional requirements for JMS in Trade 6 be clear.

Trade 6 integrates both queue based and publish/subscribe MDBs.

Point-to-point

The queue based TradeBrokerMDB asynchronously processes stock purchase and sell requests. In the Trade 6 configuration, if the Order Process mode is set to Asynchronous_1-Phase or Asynchronous_2-Phase then instead of an order for some stock being processed immediately, it is handled asynchronously. Upon buying some stock an order is opened in the Trade database and a message sent to a queue. The application then immediately returns to the client. In the background the message is picked up by the TradeBrokerMDB and processed, updating the records in the database to set the order to be completed. The next time the client returns to the Web site, they receive a notification that the order completed successfully.

This is using JMS for asynchronous communication within the application. Some important points to note on the influence the functionality has on the topology are:

- ▶ To use the default messaging provider for Trade 6, you need to configure the data store for the messaging engine(s).
- ▶ Any cluster member can process the order messages as they update the database. This means that if the client is on TradeServer1 and the message is processed by TradeServer2, the client still receives the notification. This is asynchronous communication with no need for affinity.
- ▶ The application code is the same whether you are using the default messaging provider or WebSphere MQ queue managers.

Publish/subscribe

The pub/sub based TradeStreamerMDB subscribes to a message topic called TradeStreamerTopic. Quote data for individual stocks is published to this topic as prices change, providing a real-time streaming quote feature in Trade 6. The prices change every time some stock is bought or sold.

Some important points to note on the influence the functionality has on the topology are:

- ▶ If the Trade 6 application is clustered, then each application server needs to get the publications on price changes. If this does not occur then the prices will not be kept in synchronization across the cluster. With the default

messaging provider, every subscriber receives the message no matter where the messaging engines are located.

- By default, the MDB is using non-durable subscriptions. This avoids any restrictions imposed from using durable subscriptions.

Now that we understand how Trade 6 uses JMS we move on to apply some of the guidelines described in this chapter and create some topologies.

12.8.2 Example 1: One messaging engine on the bus

To make this example realistic some basic non-functional requirements have been set. These requirements need to be specified as it is important to recognize that this example topology is not the *best* topology to use when using the default messaging provider, nor is it the best one for Trade 6. It is one of many that could provide what is needed. Each topology benefits certain criteria and needs to be chosen based on those as well as function.

The requirements are:

- Performance is important and availability is not a requirement. It is acceptable to have an outage. However, when the system is available it must perform to a high standard.
- Hardware restriction. The cost of the hardware for this application needs to be kept to a minimum.

There would normally be many other influences in choosing the topology, such as workload patterns, but this is just to give some real world justification for this example. The selected topology is shown in Figure 12-6 on page 675.

Example 1: Default messaging provider topology

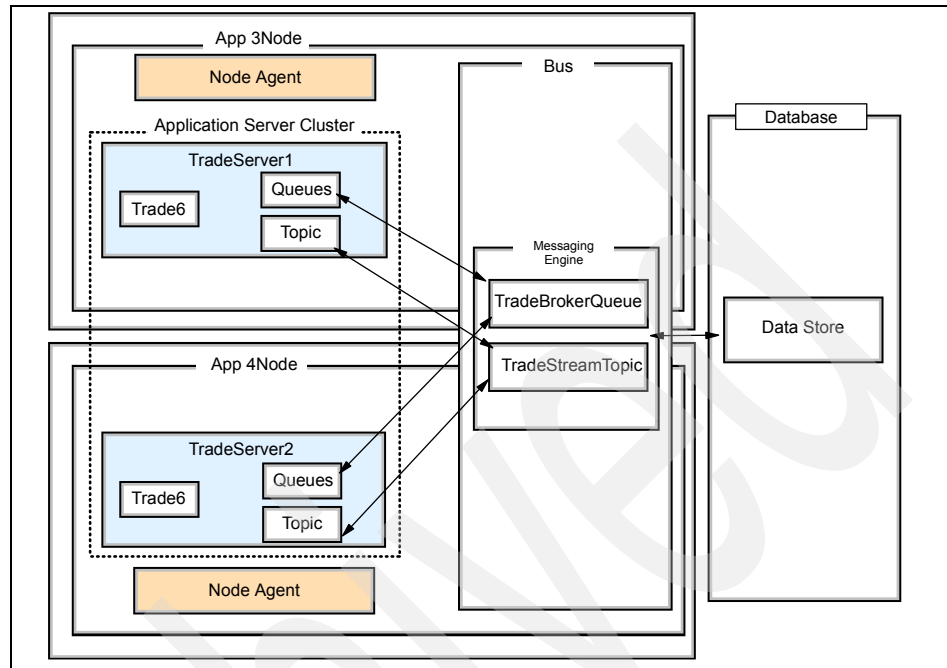


Figure 12-6 Example 1 topology for Trade 6 and default messaging provider

Key points about this topology:

- ▶ It is low cost. Only two physical machines are needed, one node can even be on the same machine as the Deployment Manager.
- ▶ The preferred server for the messaging engine is TradeServer1. The messaging engine starts on the TradeServer1 application server, however the policy configuration in this example does not require that the messaging engine run on TradeServer1 whenever possible. For example, if TradeServer1 fails and the messaging engine fails over to TradeServer2, it will stay on TradeServer2 even when TradeServer1 becomes available again.
- ▶ There are several possible points of failure in this topology. The messaging engine fails if the connection to the data store is broken, the data store stops, or the database fails. Additionally, the preferred application server for the messaging engine is a point of failure. However, if the preferred server fails, any running transactions are rolled back and the messaging engine attempts to start on TradeServer2.
- ▶ *Only one server will consume messages.* Although the Web container of each server produces messages, only the EJB container of the server local to the messaging engine (TradeServer1) consumes messages. Messages that

arrive on a local queue partition are always consumed by the local application server. If consumption of JMS messages by both EJB containers is required, the topology of one messaging engine per application server is recommended (as described in “Example 2: One messaging engine per server” on page 686). Additionally, the Web container layer and EJB container layer can be split into separate clusters (requiring two additional application servers). In this case, since the messages are all remote to the EJB cluster, the consumption of messages will be load balanced.

Note: The separation of Web and EJB containers is not recommended for performance and scalability.

This topology is fairly straight forward to configure because the Trade 6 example comes with a script that can do most of the work for you. This setup however is not the same as what is described in Chapter 8, “Implementing the sample topology” on page 387.

It involves:

1. Setting up the WebSphere Application Server cluster
2. Configuring all the JDBC and JMS resources
3. Installing the application
4. Testing the application

It is assumed that a WebSphere Application Server cell with the Deployment Manager on app3 is installed and configured. There are two application server profiles in the cell, one resides on app3 and the other one resides on app4. The nodes app3 and app4 should be federated into the cell. Once this has been done you are ready to carry on.

Note: Trade 6 can be set up with Oracle and DB2 as the underlying database. There are different requirements for the different databases and data source configuration. For example, if you use a remote IBM DB2 UDB 8.2 and DB2 Legacy CLI-based Type 2 JDBC driver, then you need to install and configure the IBM DB2 UDB Client 8.2 on the application server machine (but Trade 6 uses by default the DB2 universal JDBC Type 4 driver so you do not need to setup DB2 clients).

Example 1: Scripted installation and configuration

The Trade 6 package comes with a wsadmin script called trade.jacl that allows you to install the application and it also automatically sets up all required resources such as DB2/Oracle data sources and JMS resources. Installing Trade 6 is covered in further detail in section 8.8, “Installing and configuring Trade 6” on page 436.

The command to start the configuration script is:

```
wsadmin -f trade.jacl [configure|install|all]
```

Be sure to run `setupCmdLine.bat` first from the bin directory of your WebSphere installation. By default, the “all” mode is assumed which creates both the configuration of resources and the installation of the application.

To simply create the cluster and install the necessary JDBC/JMS resources without installing the application, append the “configure” option to the command. To install the application without creating the resources, append the “install” option to the command.

If installation of the application through the Administrative Console is preferred, run the script in “configure” mode first, then install the application manually.

Before starting the configuration script, you should make sure that the Deployment Manager has been started and all the required nodes have been federated successfully.

After the script is started, you just need to follow the directions and answer the questions as follows:

1. Specify whether global security is enabled in your environment (it is not enabled by default).

```
-----  
Trade Install/Configuration Script
```

```
Operation:  all  
Silent:     false  
-----
```

```
Global security is (or will be) enabled (true|false) [false]:
```

2. Answer **yes** to install the trade application on a cluster.

```
Is this a cluster installation (yes|no) [no]:
```

3. Answer **yes** after you have confirmed that the following requirements have been met.

```
-----  
Collecting Cluster and Cluster Member Info
```

```
Note: Before proceeding, all nodes intended for  
use in this cluster must be federated with the  
deployment manager using the addNode command!  
To ensure that this process goes smoothly, it  
is also important to verify that each node can  
ping the other cluster nodes based on the host
```

names configured within the WebSphere profile
creation tool.

Have all nodes been federated and network connectivity
verified? (yes|no) [yes]:

4. Specify the cluster name here. Examples 1 and 2 use the default name
“**TradeCluster**” and refer to the Trade 6 cluster using this name.

Please enter the cluster name [TradeCluster]:

5. Select the nodes to be added to the cluster. Make sure you enter the name of
an application server node, such as **app3Node**, and not the Deployment
Manager node from the list of available nodes.

Available Nodes:

app3Node
app4Node
dmJMSNode

Select the desired node [app3Node]:

app3Node

6. Enter the name of the cluster member server on the node selected in step 5.

Please enter the cluster member name [TradeServer1]:

Current Cluster Nodes and Members:

app3Node - TradeServer1

7. Answer **yes** if you need to repeat steps 5 and 6 to add additional cluster
members.

Add more cluster members (yes|no) [yes]:

8. After specifying the configuration for the cluster and member servers, you
need to enter the information for the datasource, **db2** is used in our example.

Collecting Database/Datasource Information

Select the backend database type (db2|oracle) [db2]:

NOTE: wsadmin requires ";" for delimiting the database
driver path regardless of platform!

Please enter the database driver path

[c:/sqllib/java/db2jcc.jar;c:/sqllib/java/db2jcc_license_cu.jar;c:/sqlli
b/java/db2jcc_licens
e_cisuz.jar]:


```
${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar;${DB2UNIVERSAL_JDBC_DRIVER_P  
ATH}/db2jcc_license_cu.jar
```

Please enter the database name [tradedb]:

Please enter the DB2 database hostname [localhost]:

Please enter the DB2 database port number [50000]:

Please enter the database username [db2admin]:

Please enter the database password [password]:

9. Once the above required information is entered, the script starts to set up the environment for the application automatically.

Note that you do not have to specify any configuration information for the default messaging provider because the script takes the default values and sets up JMS resources as follows:

```
-----  
Configuring JMS Resources  
Scope: TradeCell(cells/TradeCell|cell.xml#Cell_1)  
-----
```

Creating JAAS AuthData TradeOSUserIDAuthData...

Alias Name: TradeOSUserIDAuthData

User: LocalOSUserID

Password: password

TradeOSUserIDAuthData created successfully!

Creating SIBus TradeCluster...

TradeCluster created successfully!

Adding SIBus member TradeCluster...

Default DataSource: false

Datasource JNDI Name: jdbc/MEDataSource

SIBus member added successfully!

Creating SIB Messaging Engine...

Bus Name: TradeCluster

Default DataSource: false

Datasource JNDI Name: jdbc/MEDataSource

Cluster Name: TradeCluster

created successfully!

Creating OneOfNPOLICY Policy for ME0...

Alive Period(s): 30

Server Name: TradeServer1

ME Name: TradeCluster.000-TradeCluster
Policy for ME0 created successfully!

Modifying ME DataStore parameters...

ME Name: TradeCluster.000-TradeCluster
AuthAlias: TradeDataSourceAuthData
Schema Name: IBMME0

TradeCluster.000-TradeCluster data store modified succes

Creating OneOfNPOLICY Policy for ME1...

Alive Period(s): 30
Server Name: TradeServer2
ME Name: TradeCluster.001-TradeCluster

Policy for ME1 created successfully!

Modifying ME DataStore parameters...

ME Name: TradeCluster.001-TradeCluster
AuthAlias: TradeDataSourceAuthData
Schema Name: IBMME1

TradeCluster.001-TradeCluster data store modified succes

Creating SIB Destination TradeBrokerJSD...

Destination Name: TradeBrokerJSD
Destination Type: Queue
Reliability: EXPRESS_NONPERSISTENT
Cluster Name: TradeCluster

TradeBrokerJSD created successfully!

Creating SIB Destination Trade.Topic.Space...

Destination Name: Trade.Topic.Space
Destination Type: TopicSpace
Reliability: EXPRESS_NONPERSISTENT

Trade.Topic.Space created successfully!

Creating JMS Queue Connection Factory TradeBrokerQCF...

Connection Factory Name: TradeBrokerQCF
Connection Factory Type: Queue
JNDI Name: jms/TradeBrokerQCF

TradeBrokerQCF created successfully!

Creating JMS Topic Connection Factory TradeStreamerTCF..

Connection Factory Name: TradeStreamerTCF
Connection Factory Type: Topic
JNDI Name: jms/TradeStreamerTCF

TradeStreamerTCF created successfully!

Creating JMS Queue TradeBrokerQueue...

Queue Name: TradeBrokerQueue
JNDI Name: jms/TradeBrokerQueue

```
SIB Destination: TradeBrokerJSD
Delivery Mode:   NonPersistent
TradeBrokerQueue created successfully!
```

```
Creating JMS Topic TradeStreamerTopic...
Topic Name:      TradeStreamerTopic
JNDI Name:       jms/TradeStreamerTopic
Topic Space:     Trade.Topic.Space
Delivery Mode:   NonPersistent
TradeStreamerTopic created successfully!
```

```
Creating MDB Activation Spec TradeBrokerMDB...
MDB Activation Spec Name: TradeBrokerMDB
JNDI Name:               eis/TradeBrokerMDB
JMS Destination JNDI Name: jms/TradeBrokerQueue
Destination Type:        javax.jms.Queue
TradeBrokerMDB created successfully!
```

```
Creating MDB Activation Spec TradeStreamerMDB...
MDB Activation Spec Name: TradeStreamerMDB
JNDI Name:               eis/TradeStreamerMDB
JMS Destination JNDI Name: jms/TradeStreamerTopic
Destination Type:        javax.jms.Topic
TradeStreamerMDB created successfully!
```

```
-----
JMS Resource Configuration Completed!!!
-----
```

Tip: You may have noticed that the messaging engines are named TradeCluster.xxx-TradeCluster. The format for a messaging engine name is <cluster_name.xxx-bus_name> indicating that in this example the cluster name is the same as the bus name. You may want to employ a more meaningful bus name in a production environment.

10. The Trade 6 application is installed successfully when the following message is displayed. However, this does not guarantee that the entire script has been successful. Be sure to review each step of the output carefully for errors.

```
ADMA5013I: Application trade installed successfully.
Install completed successfully!
```

```
-----
Trade Installation Completed!!!
-----
```

```
Saving...
```

```
Saving config...
```

Example 1: Additional configuration

The following steps are necessary to further configure Trade 6 for this example:

Important: By default, the Trade 6 installation script installs one messaging engine per application server rather than a single messaging engine for the cluster. This example changes the Trade 6 default configuration manually using the Administrative Console.

1. The Trade 6 wsadmin script creates one messaging engine per application server. Remove the messaging engine corresponding to TradeServer2 in the Administrative Console under **Service Integration -> Buses -> TradeCluster -> Bus members -> Messaging engines**.
2. Remove the policy for ME1 under **Servers -> Core groups -> Core group settings -> DefaultCoreGroup -> Policies**.
3. Select **Policy for ME0** and uncheck **Fail back** and **Preferred servers only**. Using this configuration the messaging engine can be started on application server TradeServer2 in the event of a failure. Fail back enabled moves the messaging engine back to its preferred server whenever possible. The example 1 configuration assumes that running a messaging engine on TradeServer2 is acceptable even when both servers are available.

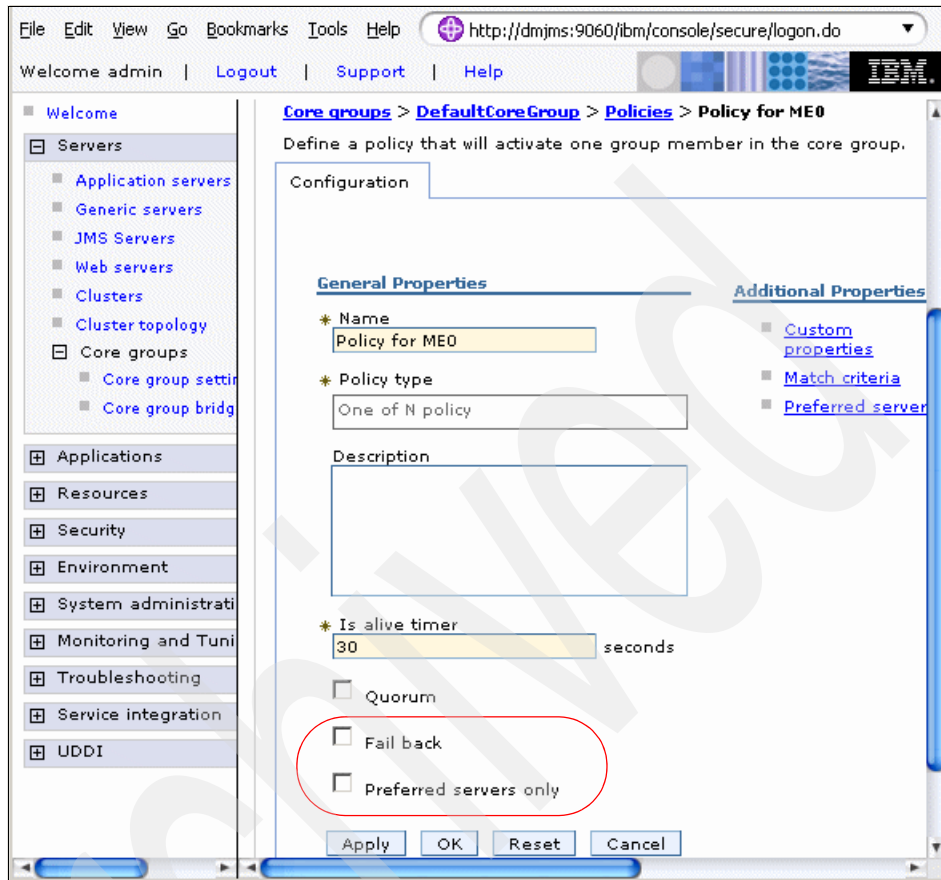


Figure 12-7 Policy for messaging engine in example 1

4. Next, start the cluster and populate the Trade 6 database (this is the application database which has a different schema from the messaging engine data stores). The main page for Trade 6 can be accessed from an HTTP or application server address:

`http://<server_name>:<port>/trade/`

On the main page, select **Configuration -> (Re)-populate Trade Database**.

Note: If the database population fails, you may need to declare `${DB2UNIVERSAL_JDBC_DRIVER_PATH}` at the node scope level under **Environment -> WebSphere Variables** in the Administrative Console. Additionally, you may want to test the database connection under **Resources -> JDBC Providers -> DB2 Universal JDBC Driver Provider (XA) -> Data sources**.

5. By default, Trade 6 uses a synchronous order processing mechanism on each server. The Trade 6 configuration can be changed to use MDBs and JMS. For each application server go to:

`http://<application_server_name>:<port>/trade/config`

and change the following configuration parameters:

- Order processing mode = Asynchronous_2-Phase
- Enable Operational Trace = True
- Enable Full Trace = True

Click **Update config** to complete the changes.

Note: The Trade 6 runtime parameters are not persisted. Thus, if you restart the Trade 6 application, you need to set them again.

Verifying the example 1 messaging configuration

This section walks through the different components configured for the default messaging provider after installation and configuration is complete:

1. The queue connection factory, topic connection factory, TradeBrokerQueue definition, TradeStreamTopic definition, and activation specification for Trade 6 can be inspected under **Resources -> JMS Providers -> Default messaging**. Be sure the scope is at the cell level.
2. Bus members (in this case the only bus member relevant to this example is the TradeCluster), messaging engines, and bus destinations can be inspected under **Service integration -> Buses -> TradeCluster**.
3. Verify that the messaging engine is started. The data store settings for the messaging engine under **Service integration -> Buses -> TradeCluster -> Messaging engines -> TradeCluster.000-TradeCluster -> Data store** should also be inspected. Notice that the schema name for the data store is IBMME0. That means that you need to set up the database correspondingly. In Oracle, you need to create a user account. For this example, create a user account called IBMME0. In DB2, the user needs the right to create a new schema called IBMME0, otherwise you need to create a new user also.

Note: Connectivity to the datastore can be tested in the Administrative Console under **Resources -> JDBC providers -> DB2 Universal JDBC Driver Provider -> Data sources -> MEDataSource**. Be sure you are at the cell scope when selecting the driver provider.

Testing the example 1 configuration

You can test the asynchronous messaging of Trade 6 by purchasing 100 shares of stock s:0 while accessing the Trade 6 Web site from the TradeServer1 Web container.

The trace log found in the profiles/logs/<application_server_name> directory on the TradeServer1 system provides insight into the workflow of the messaging configuration. As shown in Example 12-1, you can verify that the TradeBrokerMDB on the application server local to the messaging engine receives and processes the order whether you access Trade 6 from the Web container of TradeServer1 or TradeServer2.

Attention: The consumption of messages by TradeBrokerMDB is only done on the server local to the messaging engine which is TradeServer1 in this example (unless the messaging engine is moved due to a failure). This means the server local to the messaging engine will take on 100% of the message consumption load for stock sales and purchases. See 12.5, “Clustering, high availability and workload management” on page 656 for more information. Example 2 configures one messaging engine per application server which can facilitate balanced workload for message consumption.

Example 12-1 trace.log snippet verifying order processing

```
[2/15/05 11:02:24:711 CET] 00000018 SystemOut      0    TradeLog:Tue Feb 15
11:02:24 CET 2005-----

[2/15/05 11:02:24:711 CET] 00000018 SystemOut      0    TradeBroker:onMessage --
received message -->neworder: orderID=9000 runtimeMode=EJB
twoPhase=truecommand-->neworder<-- threadID=Thread[Default : 2,5,main]

[2/15/05 11:02:24:711 CET] 00000018 SystemOut      0    TradeLog:Tue Feb 15
11:02:24 CET 2005-----

[2/15/05 11:02:24:711 CET] 00000018 SystemOut      0    TradeBrokerMDB:onMessage
- completing order 9000 twoPhase=true direct=false threadID=Thread[Default :
2,5,main]
```

As shown in Example 12-2, both server trace logs should show an update in the price of stock s:0 via the TradeStreamerMDB since they are both subscribed to the TradeStreamerTopic.

Example 12-2 trace.log snippet verifying a stock price update

```
[2/15/05 11:01:20:889 CET] 00000016 SystemOut      0    TradeLog:Tue Feb 15
11:01:20 CET 2005-----
```

```
[2/15/05 11:01:20:889 CET] 00000016 SystemOut      0    TradeStreamer:onMessage  
-- received message -->Update Stock price for s:0 old price = 39.50 new price =  
56.09command-->updateQuote<-- threadID=Thread[Default : 0,5,main]
```

12.8.3 Example 2: One messaging engine per server

In this example, Trade 6 uses one messaging engine per server. The messaging engine policies are setup such that each messaging engine receives JMS messages from its corresponding application server. Unlike example 1, there is no longer a single point of failure and possible bottleneck associated with a single messaging engine on the bus. Thus, example 2 facilitates performance and scalability.

To make this example realistic some basic non-functional requirements have been set.

The requirements are:

- ▶ Performance is more important than availability. The application does not have to be available for use continuously. Outages occurring will be tolerated. However, when the system is available it must perform according to a high standard.
- ▶ Hardware restriction. The cost of the hardware for this application needs to be kept to a minimum.

The selected topology is shown in Figure 12-8 on page 687.

Example 2: Default messaging provider topology

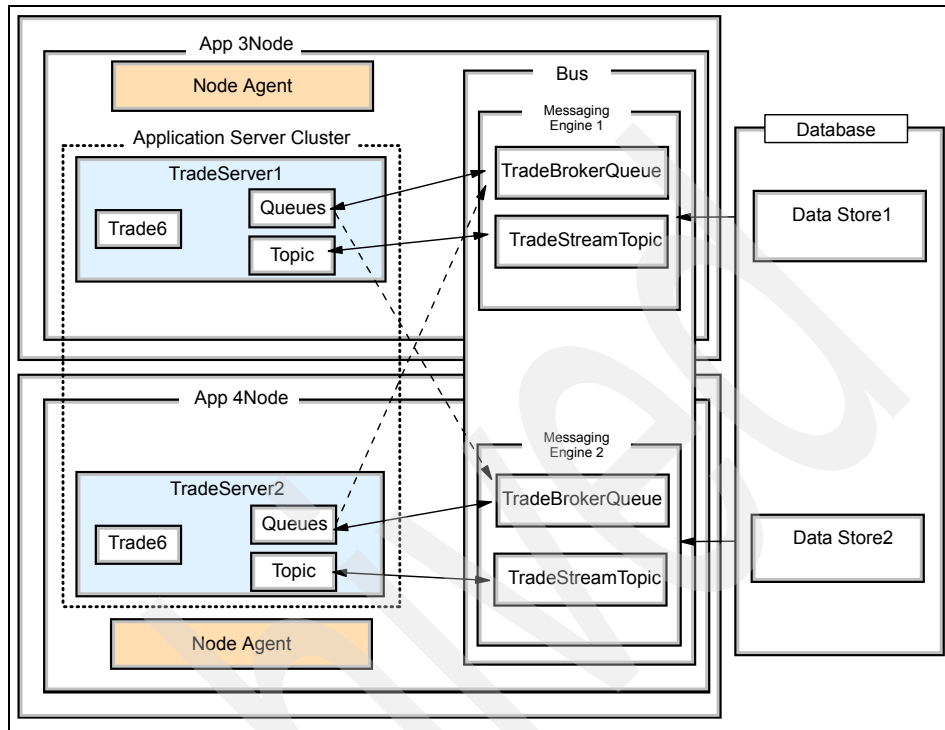


Figure 12-8 Example 2 topology for Trade 6 and default messaging provider

Key points about this topology:

- ▶ This topology uses two application servers; however, setting up more application servers as members of the cluster with their respective messaging engines should be straight forward after reviewing this example.
- ▶ Messaging workload is spread across the messaging engines. The TradeBrokerQueue is defined on the bus with the cluster as a single bus member, so the queue is partitioned between the messaging engine on each server, for more information about partitioned queues, please refer to 12.5.3, “Partitioned queues” on page 657.
- ▶ As mentioned in “Data stores” on page 649, each messaging engine uses its own data store (schema), but normally on the same database. A complete failure of the database affects both messaging engines. However, a failure of a single data store (for example due to its port being blocked) affects only the dependent messaging engine since subsequent messages are directed to the other messaging engine. Failure of one of the messaging engines is

sustainable since the remaining messaging engine would process all incoming messages while a restart is attempted.

- There is a one-to-one relationship with the load on an application server and its assigned messaging engine. If a high load is incurred on a particular application server, then, in general, you can also expect a high load on its respective messaging engine and EJB container. This is because the default behavior of the SIB JMS Resource Adapter is to connect to a local messaging engine whenever possible to increase performance.

For example, when a message is produced in the Web container of TradeServer1 it is put on the local partitioned queue which is on Messaging Engine 1 or TradeCluster.000-TradeCluster. The message is then consumed by a MDB, such as TradeBrokerMDB, by the EJB container of TradeServer1. The behavior of using a local messaging engine when producing a message and the local application server when consuming a message is the default and recommended behavior of an application running on a cluster which is a member of the bus. See 12.5.4, “JMS clients connecting into a cluster of messaging engines” on page 659.

Note: Although there is a one-to-one relationship between workload for messaging engines and respective application servers for Trade 6, this is not the case for all applications. For example, an application using the example 2 topology could create a single `javax.jms.Connection` and use it for all message production, in effect using the same SIB JMS Resource Adapter connection to one of the messaging engines. In this case, the same messaging engine is used for 100% of message production and consumption. Additionally, only the EJB container of one application server is used in message consumption. This is not recommended as the remaining messaging engine is not utilized properly. See 12.5, “Clustering, high availability and workload management” on page 656.

This topology can be setup using the Trade 6 script similar to example 1. This setup is similar but not the same as the setup described in Chapter 8, “Implementing the sample topology” on page 387.

It involves:

1. Setting up the WebSphere Application Server cluster
2. Configuring all the JDBC and JMS resources
3. Installing the application
4. Testing the application

It is assumed that a WebSphere Application Server cell with the Deployment Manager on node app3 is installed and configured. There are two application server profiles in the cell, one resides on app3 and the other one resides on

app4. The nodes app3 and app4 should be federated into the cell. Once this has been done you are ready to carry on.

Note: Trade 6 can be set up with Oracle and DB2 as the underlying database. There are different requirements for the different databases and data source configuration. For example, if you use a remote IBM DB2 UDB 8.2 and DB2 Legacy CLI-based Type 2 JDBC driver, then you need to install and configure the IBM DB2 UDB Client 8.2 on the application server machine.

Example 2: Scripted installation and configuration

The installation of Trade 6 for example 2 should be done exactly as described for example 1. See “Example 1: Scripted installation and configuration” on page 676.

Example 2: Additional configuration

The following steps should be used to further configure Trade 6 for this example:

1. Next, start the application and populate the Trade 6 database (this is the application database which has a different schema from the messaging engine data stores). The main page for Trade 6 can be accessed from an HTTP or application server from:

`http://<server_name>:<port>/trade/`

In the main page, select **Configuration -> (Re)-populate Trade Database**.

Note: If the database population fails you may need to declare `${DB2UNIVERSAL_JDBC_DRIVER_PATH}` at the node scope level under **Environment -> WebSphere Variables** in the Administrative Console. Additionally, you may want to test the database connection under **Resources -> JDBC Providers -> DB2 Universal JDBC Driver Provider (XA) -> Data sources**.

2. By default, Trade 6 uses a synchronous order processing mechanism on each server. The Trade 6 run-time configuration can be changed to use MDBs and JMS. For each application server go to:

`http://<application_server_name>:<port>/trade/config`

and change the following configuration parameters:

- Order processing mode = Asynchronous_2-Phase
- Enable Operational Trace = True
- Enable Full Trace = True

Click **Update config** to complete the changes.

Verifying the example 2 messaging configuration

This section walks through the different components configured for the default messaging provider after installation and configuration is complete:

1. The queue connection factory, topic connection factory, TradeBrokerQueue definition, TradeStreamerTopic definition, and activation specification for Trade 6 can be inspected under **Resources -> JMS Providers -> Default messaging**. Be sure the cell scope is at the cell level.
2. The bus members (in this case the only bus member relevant to this example is the TradeCluster), messaging engines, and bus destinations can be inspected under **Service integration -> Buses -> TradeCluster**.
3. Verify that both messaging engines are started. The data store settings of each messaging engine under **Service integration -> Buses -> TradeCluster -> Messaging engines -> TradeCluster.000-TradeCluster -> Data store** in the Administrative Console should also be inspected. Notice that the schema name for each data store as defined by the Trade 6 wsadmin script is different (IBMME0, IBMME1, and so on). That means that you need to set up the database correspondingly. In Oracle, to have different schemas, you need to create different user accounts. For this example, create user accounts called IBMME0, IBMME1, and so on. In DB2, as long as the user has the right to create a new schema, multiple schemas can be created.
4. The policies for the messaging engines can be found under **Servers -> Core groups -> Core group settings -> DefaultCoreGroup -> Policies**. Notice that **Preferred servers only** is selected and one distinct application server is included in the match criteria for each messaging engine. For more information about policies and match criteria see section 12.5.5, “Preferred servers and core group policies” on page 660.

Testing the example 2 configuration

Reviewing the trace logs for example 2 can be done exactly as described in “Testing the example 1 configuration” on page 685.

It should be observed that the consumption of a message in this example is always on the application server which produced the message unlike the example 1 configuration where only one application server consumes messages. Thus, if you access Trade 6 from the Web container of TradeServer2 and make an order, the order message should be produced and consumed from TradeServer2.

12.9 Workload management example using BeenThere

BeenThere is a sample J2EE application which facilitates monitoring of workload management among application servers and is one of the sample applications provided with WebSphere V6. However, there is a version of BeenThere available that has been updated to use the default messaging provider and the bus for WebSphere Application Server V6.0. Extensive documentation and the updated BeenThere.ear is provided in the repository for this redbook. Refer to Appendix B, “Additional material” on page 1037 for information about how to obtain the modified EAR file.

BeenThere displays workload at the application server level, thus using two clusters to separate the Web and EJB containers is recommended so inspection of which EJB container consumes workload is clear. The documentation provides step-by-step directions for this configuration.

Even though the containers are split (which can adversely affect performance), a Web container always has its messages consumed by the same EJB container (unless there is a messaging engine failure), because the BeenThere code is written to use the same JMS connection for subsequent message production.

Thus, a Web container running BeenThere uses the same messaging engine it initially connects to throughout the application's lifespan. The connections to messaging engines are workload balanced in the sense that each Web container, or server in the Web container cluster, connects to a different messaging engine in the EJB cluster. For workload balancing to occur respective to HTTP requests, an application needs to create a new JMS connection for each HTTP request. See 12.5.4, “JMS clients connecting into a cluster of messaging engines” on page 659.

12.9.1 Taking advantage of the BeenThere documentation

The documentation provided with the modified BeenThere version includes a step-by-step guide for installing the application and configuring resources. It is recommended that the documented topology be setup before attempting to use any of the possible topologies described in the next section. Figure 12-9 on page 692 illustrates the documented configuration for BeenThere.

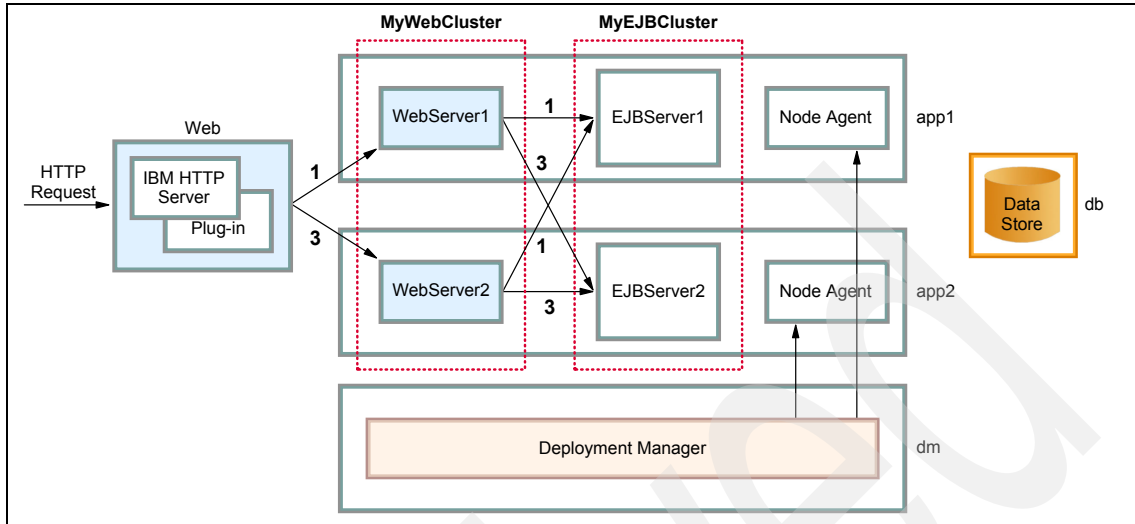


Figure 12-9 BeenThere documented topology

To access the documentation:

1. Download **BeenThereDocumentation.zip** from the redbook repository.
2. Unzip all files and folders. Be sure file paths and folders are preserved.
3. Open **index.html**.
4. Select **Configure and run**.

12.9.2 Possible topologies to use with BeenThere

When architecting an enterprise solution, it is often useful to test the messaging topology, cell topology, network security configuration, and any other environment related decisions with a simple application before testing or even designing a new enterprise messaging application. BeenThere is an ideal application for this task as its sole purpose is to analyze workload on Web and EJB containers.

It also may be used to learn about possible topologies and the default messaging provider in general. Here are some topologies to try for workload analysis with BeenThere:

- One messaging engine per EJB application server cluster setup such that only preferred servers are used.

This is the topology configured in the BeenThere documentation. Be sure to stop additional messaging engines in the Administrative Console and inspect what EJB cluster member picks up the missing engine's workload.

- One messaging engine per EJB application server cluster setup such that non-preferred servers can be used.

This topology only requires a single change from the documented configuration. Uncheck **Preferred servers only** in the Administrative Console under **Servers -> Core groups -> Core group settings -> DefaultCoreGroup -> Policies -> MyEJBServerX ME Policy** for each messaging engine. Stopping application servers is insightful in this topology, because the messaging engines can restart on non-preferred servers.

- One messaging engine per application server.

This topology adds messaging engines to the Web cluster. If the messaging stores are also local to each application server, this facilitates high performance as all messaging requests from a Web container can be handled locally. This requires repeating steps 12, “Adding a messaging engine to a cluster bus member” and 13, “Creating a Core Group Policy for a messaging engine” in the BeenThere documentation for each application server in the Web cluster. The important observation to make with this topology is that the EJB cluster will go unused for message consumption since, by default, local messaging engines are always used when possible. This default behavior can be changed by altering the Target, Target type, and Target significance settings in the connection factory. See 12.6.1, “Important connection factory settings” on page 663.

12.10 Monitoring performance with Tivoli Performance Viewer

This section assumes an understanding of how Tivoli Performance Viewer works. If you need to learn more first then go to Chapter 14, “Server-side performance and analysis tools” on page 769.

The performance monitoring service within each application server provides statistics on the following areas for applications using the default messaging provider:

- SIB service Performance Module

The SIB service Performance Module provides various counters to monitor the performance of the default messaging provider components, such as message destinations and store management in messaging engines. For example, the following counters are found under **Performance Modules -> SIB Service -> SIB Messaging Engines -> <cluster_name>.xxx-bus_name -> Destinations -> Queues**:

- Available Message Count

The number of messages available for a queue for consumption. If this number is close to the destination high messages value then review the high messages value.

- Total Messages Produced

The total number of messages produced to this queue, for the lifetime of this messaging engine.

- Total Messages Consumed

The total number of messages consumed from this queue, for the lifetime of this messaging engine.

For a complete list and description of available counters related to messaging and the bus, please look at the InfoCenter article “System Integration Bus (SIB) and Messaging counters”.

- ▶ Enterprise bean Performance Module

The enterprise bean module provides the counters to monitor the behaviors of enterprise beans. In applications involving MDBs, the average response time of method `onMessage` is a key counter. There are several other counters related to MDBs, specifically:

- MessageCount

The number of messages delivered to the bean `onMessage` method.

- MessageBackoutCount

The number of messages that failed to be delivered to the bean `onMessage` method.

- WaitTime

The average time to obtain a `ServerSession` from the pool.

- ServerSessionPoolUsage

The percentage of the server session pool in use.

- ▶ JCA Connection Pools Performance Module

The counters related to the connections for the default messaging provider are listed under **Performance Modules -> JCA Connection Pools**. You should monitor the following three counters:

- CreateCount

The total number of managed connections created.

- CloseCount

The total number of managed connections destroyed.

– AllocateCount

The total number of times that a managed connection is allocated to a client (the total is maintained across the pool, not per connection).

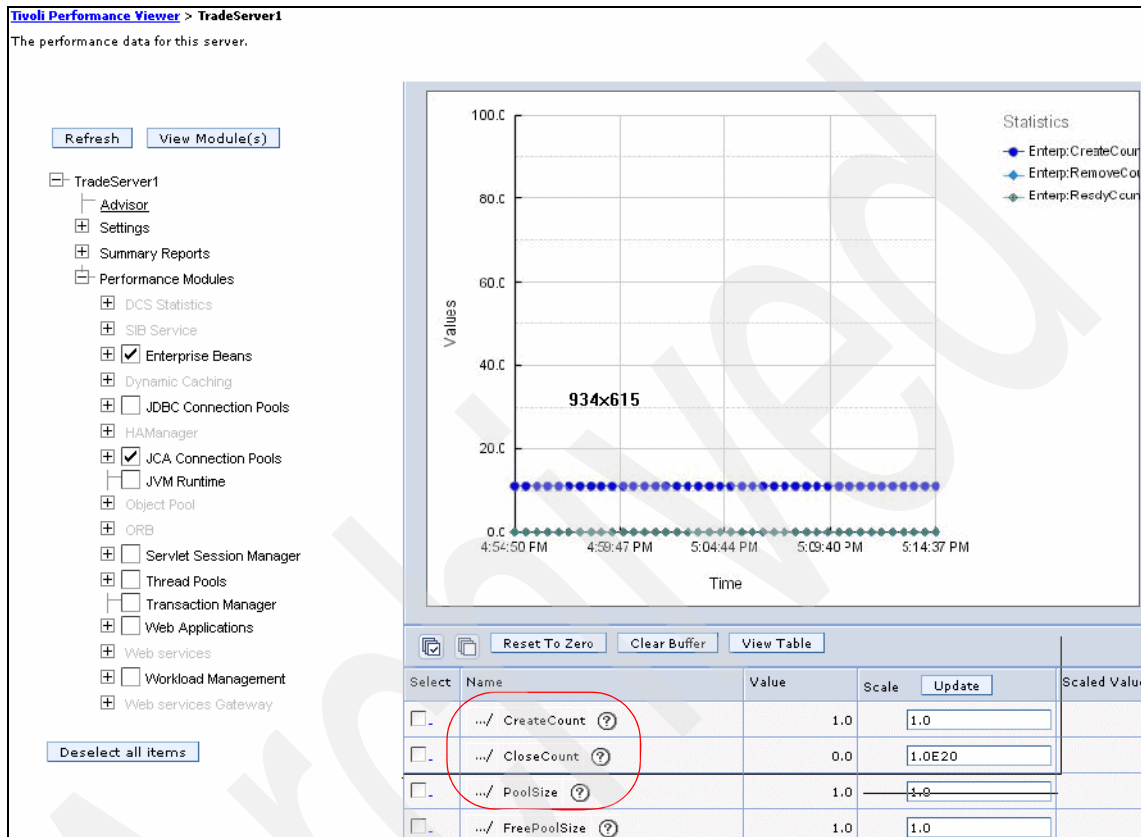


Figure 12-10 JCA Connection Pool counters

For more information about monitoring counters available to the Tivoli Performance Viewer, please refer to 14.2.1, “Performance data classification” on page 773 or read the topic “PMI data organization” in the IBM WebSphere Application Server V6 InfoCenter which can be found at:

<http://www.ibm.com/software/webservers/appserv/infocenter>

Archived

Understanding and optimizing the use of WebSphere MQ

WebSphere MQ is a market share leader in message-oriented middleware. It is implemented on numerous platforms and supports communication across nearly every popular network protocol. WebSphere MQ supports messaging for both JMS and non-JMS applications with a native API that supports all major programming languages. WebSphere MQ is used for mission critical industrial strength applications spanning every business sector.

This chapter describes how the various components of the WebSphere MQ provider interact for a JMS application and discusses optimal configuration. It takes the reader through manually configuring messaging components for the Trade 6 application and demonstrates two example topologies using Trade 6 with WebSphere MQ and WebSphere Business Integration Event Broker.

For a discussion of the default messaging provider, which is very efficient as a local messaging provider on the same node as an application, see Chapter 12, “Using and optimizing the default messaging provider” on page 643.

13.1 Introduction

Performance and stability of an application using JMS are largely governed by:

- ▶ Efficient and safe application usage of JMS
- ▶ The most efficient messaging and application server topology, but also one that provides adequate failure handling
- ▶ Optimal configuration settings of each individual application server and its resources

Stability is also important for maintaining performance, the application will need to be able to cope with failures in a graceful manner. If it is unable to do this then performance will degrade and the service provided by the application becomes unacceptable.

It is the aim of this chapter to provide the reader with enough information to cover these areas, either through discussion or by identifying other documents that already cover a particular area.

This chapter is structured so that each section builds on the knowledge from previous sections. It starts with taking a look at what happens when you use JMS in your application. This is so that the following sections can use this to help build the larger picture of the implications of making configuration changes within a JMS setup.

While some terminology and concepts are described within this chapter in completeness, others might require a previous understanding of the technology behind JMS. If you are new to JMS, then you should first look at chapter 10, “Asynchronous messaging” in the *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451 and at Chapter 11, “Using asynchronous messaging for scalability and performance” on page 621.

Also, the Trade 6 application is used in our example, so familiarity with this application allows you to better understand what is being described. For more information see the documentation that comes with the download package and 13.5.1, “What does Trade 6 use JMS for?” on page 730.

Trade 6 can be downloaded from:

<http://www.ibm.com/software/webservers/appserv/performance.html>

Note: At the time of writing this redbook, Trade 6 was not yet available for download. It is expected soon. Please monitor this page for availability.

13.1.1 JMS component diagram for sending a message

Putting a message onto a message destination using JMS requires a number of components. Figure 13-1 on page 700 depicts how the ServletToDestination servlet places a message on a destination. Since the JMS 1.1 unified API is used, the same code can be used to put a message on a topic. For an introduction to the unified JMS 1.1 API, 11.2, “Basic use of the JMS API” on page 622 provides a complementary reference.

For this example, the following are needed:

- ▶ JNDI Name service
Described in detail in Chapter 13 of the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.
- ▶ Connection Factory (CF)
Encapsulates the settings necessary to connect to a messaging system.
- ▶ Destination
A reference to the messaging destination (topic or queue).
- ▶ Queue manager
A queue manager is a WebSphere MQ term. It refers to the component that is responsible for looking after a number of queues within one install of WebSphere MQ server.
- ▶ A message destination
The actual queue or topic where messages are held until removed by an application.

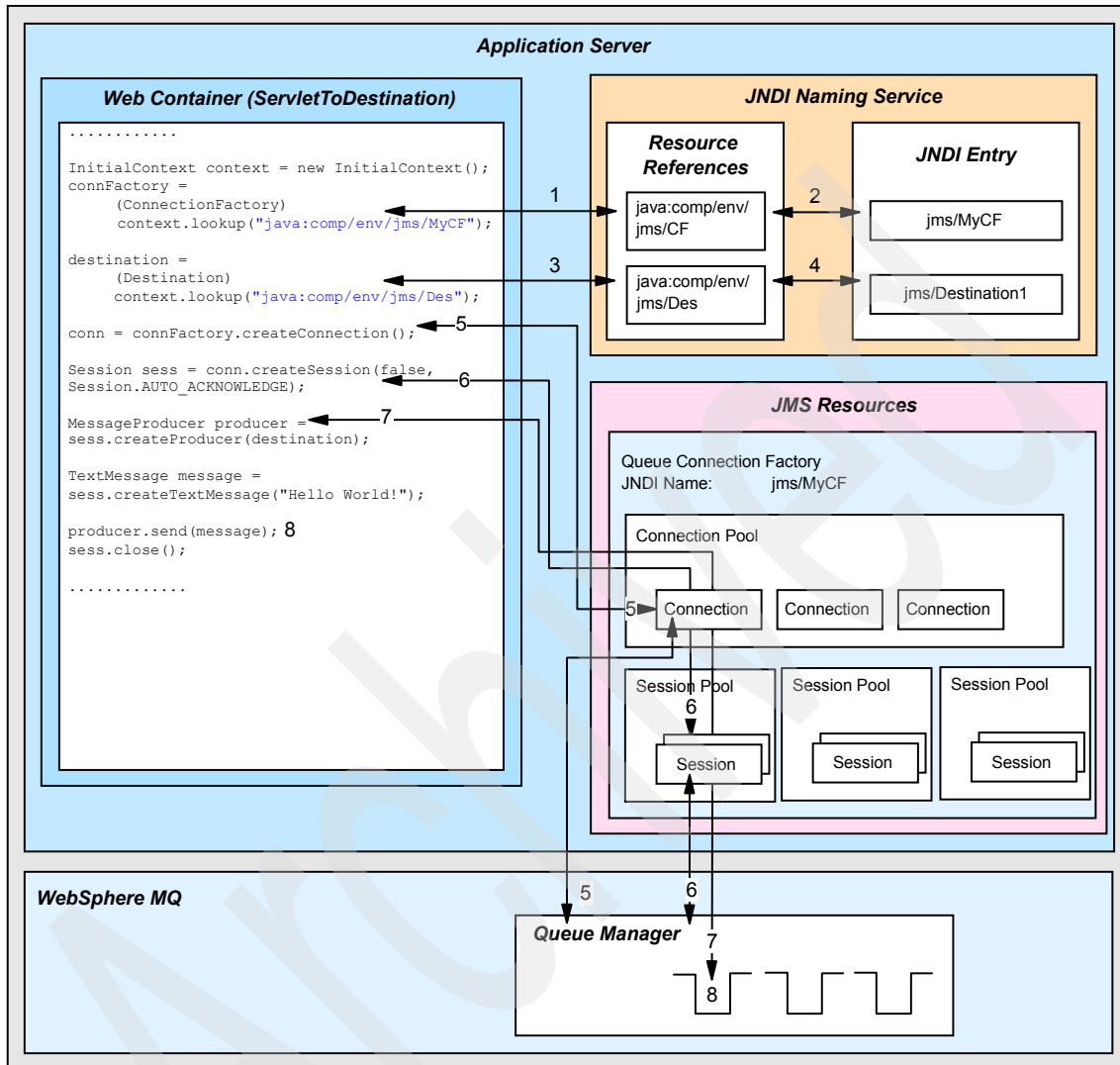


Figure 13-1 Component interactions to send a message using JMS

1. The servlet is invoked by the user and the doGet method is called by the Web container.
2. The application needs to locate the correct destination to place the message on. The JNDI namespace is used to house a link to the Java objects that will perform communication with the messaging system. The ServletToDestination servlet uses *resource references* within the code. These are linked at application deployment time to the JNDI entries that can be used to communicate with the messaging system.

3. The code performs a JNDI lookup (1) for the connection factory using the resource reference.
4. The resource reference is matched to the JNDI entry that contains the connection factory(2). The connection factory (CF) object is returned, it will be used to get a connection to the MQ queue manager that is responsible for the target destination.
5. A second JNDI lookup is performed (3) and resolved to the correct JNDI entry (4). This time it is for the destination, this will be used to locate the required queue from the queue manager.
6. To be able to communicate with the queue in the messaging system, the application must first create a connection (5) from the CF object. WebSphere Application Server maintains a pool of connection objects per CF for use by any application in the application server. This request to create a new connection will obtain an existing connection from this pool if there are any available, otherwise one will be created and added to the pool.
7. When using WebSphere MQ as the underlying messaging system, the creation of the connection will attempt a physical connection with the queue manager (5) to verify that it is available.
8. The next step is to create a session object using the createSession method (6). This is the step where a physical connection will be established to the queue manager which will eventually be used to transmit the message to the queue. WebSphere Application Server maintains a pool of session objects for each established connection on the connection factory. This request to create a new session will obtain an existing session from this pool if there are any available, otherwise one will be created and added to the pool.
9. With a connection now established to the messaging system through the session object, the application now specifies what sort of action is going to be performed; in this case, it is to send a message (7). The destination definition that was taken from the JNDI Name service is passed to the session object and this tells the queue manager the specific queue on to which the message will be placed.
10. The message is constructed and sent to the message queue (8).

This is just a basic example of how JMS might be used within an application. However, it demonstrates the number of components that are used just to send a message. The configuration of these components can be crucial in making sure that the application performs fast enough for requirements.

13.1.2 JMS and message-driven beans

Using message-driven beans incorporates further components since the application server is now responsible for delivering the message to the application.

For example, the `PingServletToMDBQueue` servlet, which is available in the Trade 6 sample application, places a message on a queue associated with the JNDI name `jms/TradeBrokerQueue`. Within the default configuration that comes with Trade 6, a listener port has been defined that monitors this queue. Any messages that arrive on the queue are passed to the `TradeBrokerMDB` that then processes them. This process is shown in Figure 13-2 on page 703.

For this example, the following are used:

- ▶ Components similar to those in 13.1.1, “JMS component diagram for sending a message” on page 699:
 - JNDI Name service
 - Queue connection factory (QCF)
 - Queue destination
 - Queue manager
 - A message queue
- ▶ Message listener service

The process within each WebSphere Application Server that is responsible for all its listener ports.
- ▶ Listener port

A listener port is WebSphere Application Server’s method of linking an MDB with the correct connection factory and destination. The information in a listener port definition is used to monitor a message queue. The JMS provider will then pass any messages that arrive on the queue to the MDB that is associated with the listener port.
- ▶ Message-driven bean (MDB)

When a message arrives on a queue that is being monitored by a listener port, the `onMessage` method of its associated MDB is called and the MDB will consume that message.

Note: This example uses the backwards compatibility of JMS 1.1 with JMS 1.0.2b. Thus, it does not use the unified API.

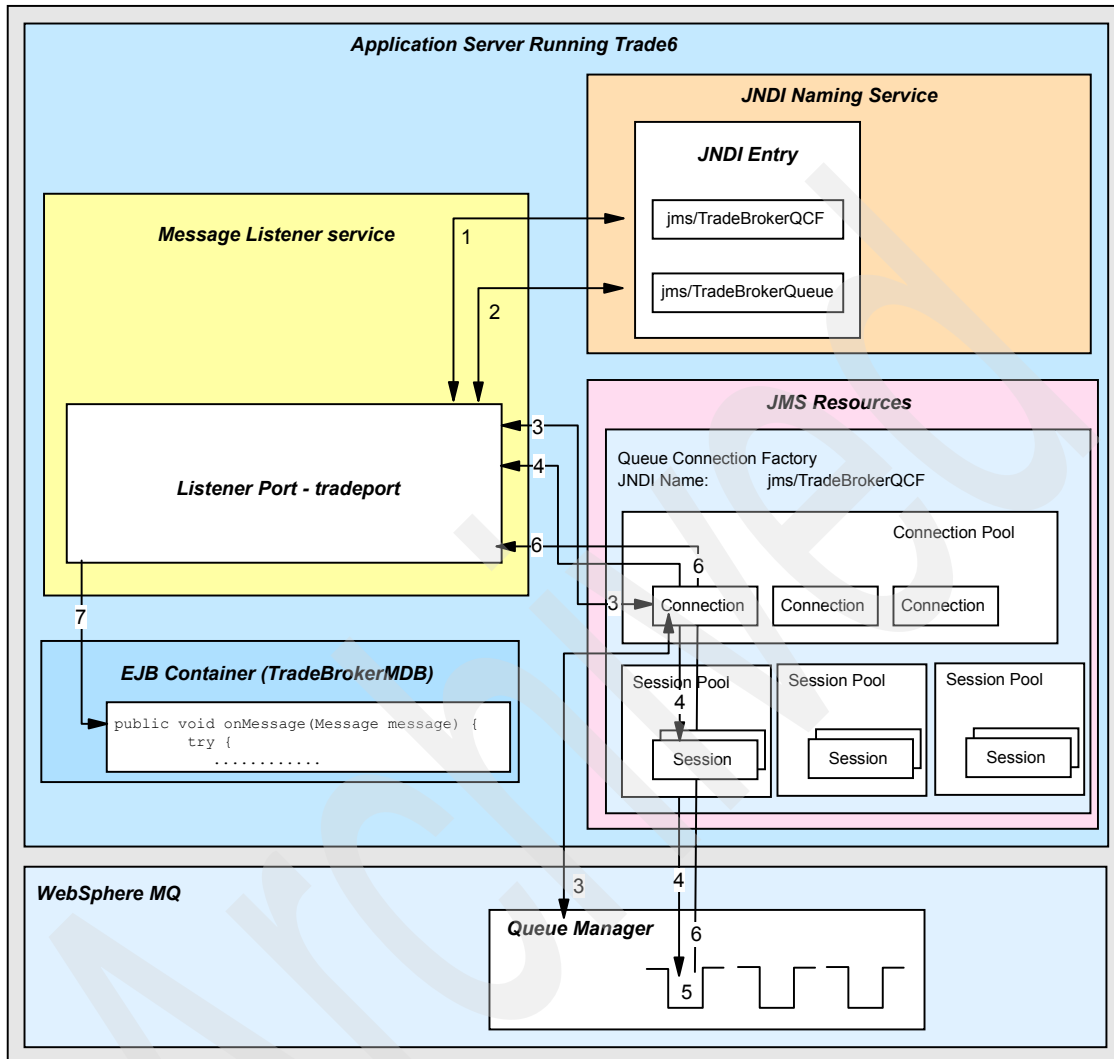


Figure 13-2 Component interactions to receive a message from a queue using a message-driven bean

To fully understand what is happening here, this example needs to be taken from the initialization point of the listener port, rather than when a message arrives. A listener port can be started and stopped independently of the application server that it is running on, so this process has specifically to do with the initialization of the listener port rather than the initialization of the application server. By default, the *initial state* of a listener port is started.

Also, Figure 13-2 does not show the detail of what is occurring inside of the message listener service. There can be multiple threads of work running inside

of the message listener service but this is not depicted; this example assumes that there is one message listener service thread that is being used by the listener port: TradePort.

1. The MDB is associated with a listener port that is in turn associated with the correct queue. Upon initialization, the listener port performs a JNDI lookup (❶) for the QCF that is specified in the listener port configuration. The QCF object is returned, it is used to get a connection to the queue manager that is responsible for the target queue.
2. The listener port does a JNDI lookup for the queue destination (❷) that it has been configured with. This is used to locate the required queue from the queue manager.
3. Using the QCF, the listener port connects to the queue manager through a QueueConnection (❸). WebSphere Application Server maintains a pool of QueueConnection objects per QCF for use by any application in the application server. This request to create a new QueueConnection will obtain an existing QueueConnection from this pool if there are any available, otherwise one will be created and added to the pool.

While the listener port is running, the QueueConnection it uses is available to any other application or listener port running with the application server. It is only when the listener port is stopped that the QueueConnection is returned to the connection pool on the QCF. Please read 11.4, “Managing workload for asynchronous messaging” on page 627 to understand more about the implications of this. This is also discussed briefly in “MQ JMS component relationships” on page 705.

4. The listener port uses the QueueConnection it has created to create a connectionConsumer object. The MQ JMS provider does the listening for messages so the provider uses the connectionConsumer to ask for listener port sessions to handle incoming messages. A listener port session can be thought of as one running thread of the listener port. It is the listener port session that calls the MDB passing it the message. Upon startup of the listener port, one listener port session is run. At the point a listener port session is created, a QueueSession (❹) is established to point at the queue the listener port is responsible for.

For ease of understanding there is only one listener port session and therefore one QueueSession object in use in our example. For more information about changing the maximum number of sessions see 11.4, “Managing workload for asynchronous messaging” on page 627 and “MQ JMS component relationships” on page 705.

This is the stage where a physical connection will be established to the queue manager which is used to monitor for the arrival of messages on the queue. WebSphere Application Server maintains a pool of QueueSession objects for each established QueueConnection. Each request to create a new

QueueSession obtains an existing QueueSession from this pool if there are any available, otherwise one is created and added to the pool.

5. The QueueSession is established and is pointing at the correct queue. The listener port is initialized. Any messages arriving on the message queue are picked up, one at a time, and processed by the correct MDB.
6. A message arrives (5) on the queue.
7. As the listener port is ready to accept messages, it takes the delivered message off the queue (6). The listener port will then pass the message on to an instance of the MDB that it is associated with. The `onMessage` method is called on the MDB (7) passing it the message.

Using MDBs requires an understanding of many different areas. The application can be designed to use MDBs, but the way in which those MDBs behave can be significantly affected by the way in which its controlling components are configured - the message listener service and listener ports. Those are in turn affected by the underlying communication mechanisms to reach the MQ JMS provider, for example the number of connections and sessions available in a connection factory.

13.2 MQ JMS component relationships

The more locations in an application that use JMS, the harder it is to optimally configure all the JMS components. There are many relationships and dependencies between the components, some of which have already been discussed above. When configuring a component in the JMS infrastructure of the WebSphere MQ provider it is essential that you understand what resources will be needed to make it work. When changing certain parameters the requirements the change places on the rest of the components is not always obvious.

13.2.1 Component relationships when using MDBs

This is especially the case when using MDBs. Figure 13-3 on page 706 shows how all the JMS components are linked together to allow an MDB to work when receiving point-to-point messages via a connection factory. This can be used to help map out the impact of making changes to the configuration.

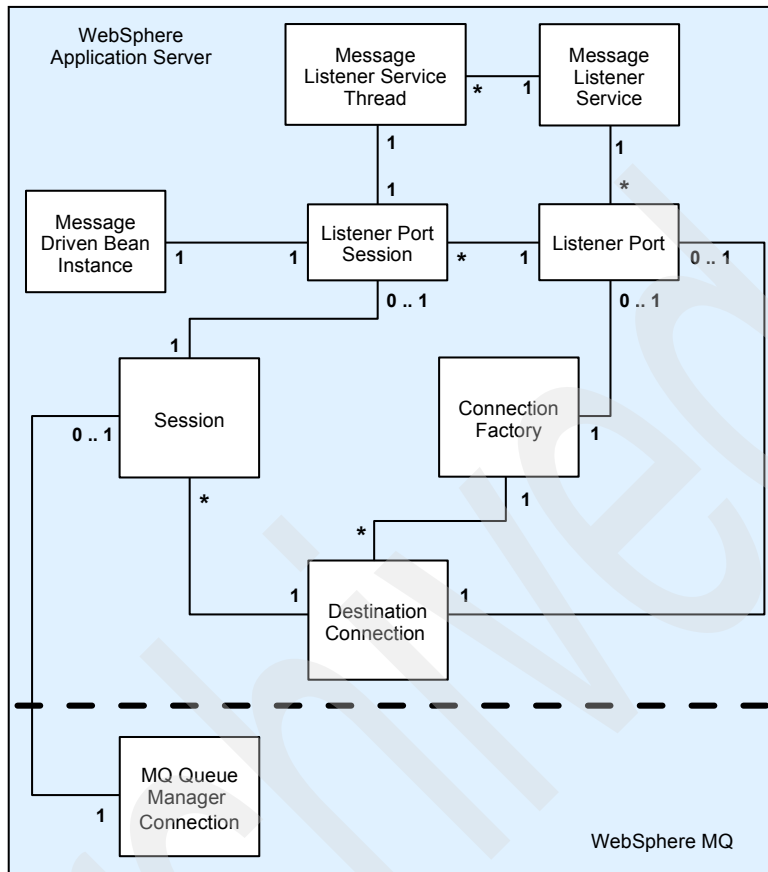


Figure 13-3 Relationships between JMS components for MDB

To read this diagram, select the component you want to understand more about and then read the connectors going from it to the other components. For example 1 session object can have 0 or 1 Listener port sessions associated with it, it will also have 1 physical queue manager connection associated with it.

Table 13-1 on page 707 and Table 13-2 on page 708 will help in interpreting Figure 13-3. Table 13-1 on page 707 describes how to interpret individual relationships.

Table 13-1 Diagram key for Figure 13-3 on page 706

Relationship	Meaning
1 to 1	Where the relationship is 1 to 1 meaning that for the first component to operate there has to be one of the second component available. Example: A listener port session requires a message listener thread. A message listener thread is always associated with a listener port session.
1 to 0 .. 1	Where the relationship is 1 to 0 .. 1 meaning that the first component can <i>optionally</i> be associated with the second component. Example: One session can have 0 or 1 listener port sessions associated with it.
0 .. 1 to 1	Reads the same as 1 to 1. See 1 to 1.
1 to *	1 to * means that component one has many instances of component two associated with it. On this diagram a 1 to * is showing control. Component one is in charge of multiple component two's. Example: The message listener service has many message listener threads.
* to 1	Reads the same as 1 to *, except that when it is a * to 1 relationship it is showing a child - parent relationship. Example: One session is managed by one connection.

Some of the boxes on the diagram represent configurable components that can be altered, changing the number of them that exist. These relationships become important when configuration changes are made as each change could have an effect on other components. Table 13-2 on page 708 shows some examples of using Figure 13-3 on page 706 to help in finding side effects.

Table 13-2 Example linked relationships for JMS components

Scenario	Requirements
Increase the number of sessions by one	As there is only a single 1 to 1 relationship for a session the only requirements to be able to do this are to make sure there are enough queue manager connections available. Remember though, by increasing the number of sessions in the session pool by one, it is affecting all the session pools for all the connections established from that connection factory. So it will in fact allow connections * 1 more sessions, not just one more session. The setting for the maximum sessions in the session pool will be determined by the connection that needs the most sessions.
Increase listener port sessions by one	Reading the diagram shows that a listener port session is reliant on a session and a message listener thread. This means that there needs to be enough of these available for it to be increased. Increasing the message listener threads has no impact on any other component, except that there need to be enough message listener threads for all listener port sessions. Increasing the sessions is described above.
Add a new listener port with one listener port session	To do this there needs to be enough message listener threads, connections and sessions available. Increasing the number of sessions will not be necessary as this new connection has its own pool of sessions, which has at least one in the pool.

13.3 Choosing optimal configuration settings

This section is a list of guidelines and settings that can help in finding the all important starting point for performance tuning an application.

Important: Each application operates differently depending on the way it uses JMS components. Any values mentioned in this section should not be treated as a generic optimal value.

13.3.1 Creation of the MQ provider objects at the correct scope

If your WebSphere Application Server application is going to be clustered across multiple application servers then the MQ queue manager has to be set up so that message sending and receiving works correctly in that cluster.

There are many different topologies that can be configured for messaging systems. A detailed discussion of these topologies can be found at:

http://www.ibm.com/developerworks/websphere/library/techarticles/0310_barci_a/barcia.html

Note: This document is currently being updated for WebSphere V6 and will become available on developerWorks in the near future.

This paper discusses some of the choices available to the infrastructure designer in choosing the optimal topology for an application to use. 13.5.2, “Clustered Trade 6 with WebSphere MQ and WebSphere Business Integration Event Broker” on page 732 demonstrates a topology that could be used in a production environment and some reasons for choosing it.

Before going into configuring pool sizes and other connection factory settings it is important that the scope level be correct. As shown in Figure 13-4 on page 710, WebSphere Application Server V6 offers five levels of scope when defining a resource that affects its visibility to applications:

- ▶ **Cell**
If a resource is defined at cell level then it will be visible to all nodes and application servers within that cell.
- ▶ **Node**
A resource defined at node level is only visible to that node and any application servers residing on that node.
- ▶ **Cluster**
Resources defined at the cluster level are visible only to servers on the named cluster. This is a new feature in WebSphere Application Server V6.0, so all cluster members must be at least at version 6.0 to use the cluster scope.
- ▶ **Server**
Resources at server level are only visible to that application server and nowhere else.
- ▶ **Application**
This is another new scope level in WebSphere Application Server V6. It limits the visibility to the named application. Note that application scope resources cannot be configured from the Administrative Console. You must use the Application Server Toolkit (AST) or the wsadmin tool to view or modify the application scope resource configuration. Resource factories that are defined

within the application scope are only available for use by this application. The application scope overrides all other scopes.

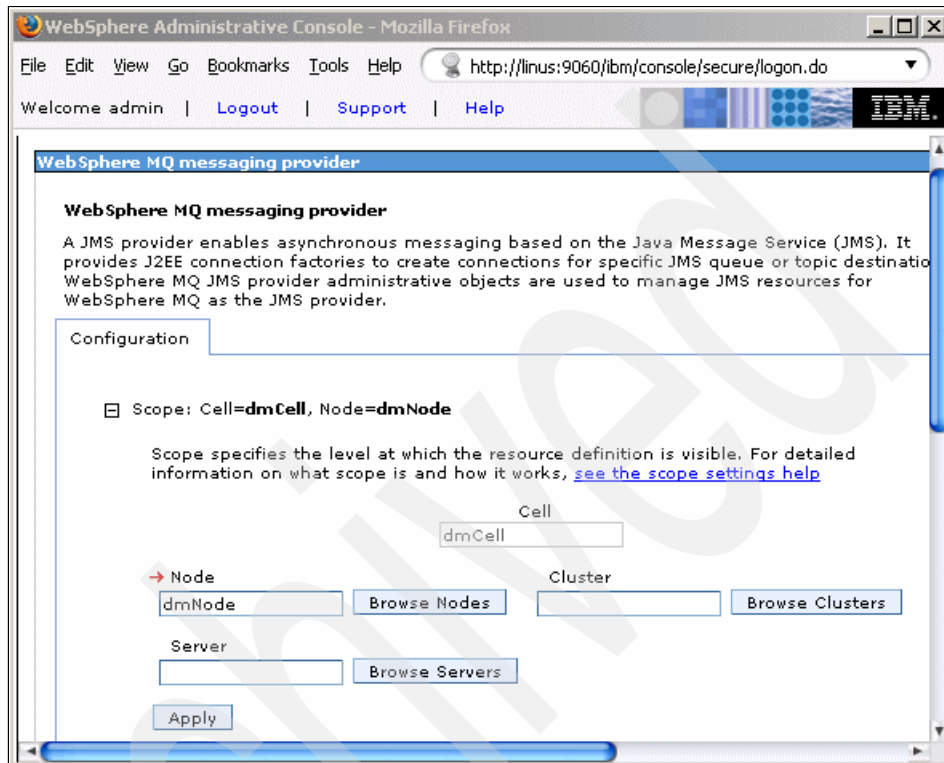


Figure 13-4 The scope level setting

Depending on the underlying messaging system topology it might be necessary to use the scope level setting to match up application servers with the correct messaging destinations and queue managers.

Defining a connection factory at the *cell* level means that *all* applications running in the cell can see this resource. When the connection factory is defined, it is the specified JNDI name that will be used by the application to find the connection factory. If the connection factory is defined at the cell level then that connection factory will be bound to that JNDI name in all processes started in the cell. Consequently any call to that JNDI name will link to the same connection factory and therefore to the underlying messaging component that is associated with that connection factory.

When using the WebSphere MQ provider, any part of an application using this connection factory, in any server in the WebSphere Application Server cluster,

would be accessing the same queue manager as is shown in Figure 13-5 on page 711.

Setting the scope level correctly not only affects how the application interacts with the messaging system but it also defines how many connection factories and TCFs you have to create and administer. If there are four nodes in your cell and the underlying topology requires that each node accesses a different queue manager, then that is going to be four connection factories created at node level.

More information and some examples on configuring connection factories at different scope levels to match the messaging topology can be found in 13.5, “Example JMS topologies and scenarios” on page 729.

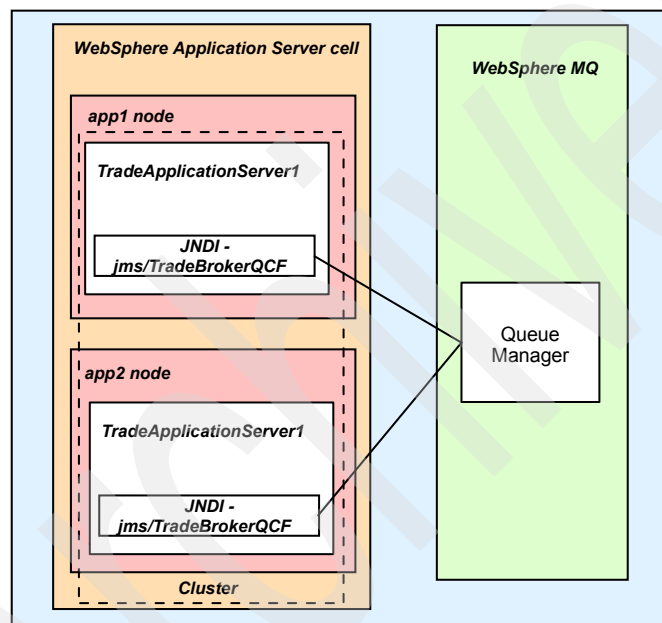


Figure 13-5 Define a resource at cell level

13.3.2 Important MQ JMS component settings

In addition to connection pool and session pool settings on a connection factory (see “Setting up the connection factory pools” on page 724) there are a number of settings that can affect performance. The settings will only work with the correct combination of messaging topology and application usage of JMS. This is specified under each heading.

Transport Type - BINDINGS or CLIENT

This setting determines the method the application server uses to communicate with the queue manager.

- ▶ Component: Connection factory
- ▶ Applies to specific messaging topologies: Yes
- ▶ Requires specific usage of JMS in application: No
- ▶ Location of setting:

Resources -> JMS Providers -> WebSphere MQ -> WebSphere MQ connection factories -> <ConnectionFactory_Name> -> Transport type field

If the queue manager is running on the same physical machine as the application server, then it is possible to set the transport type to BINDINGS. If the queue manager is on a remote machine then CLIENT has to be used. This could also affect whether transaction support is provided to MQ. To enable XA support, the queue manager must either be accessed using BINDINGS (and therefore local) or accessed using CLIENT and the specific XA enabled MQ client must be installed on the application server machine.

Using the BINDINGS transport type removes some of the overhead of remote access and is generally faster as it does not have to perform as many actions to access the queue manager.

Persistent Messages

Message persistence means that if a MQ queue manager fails, all the messages it holds will be recoverable as they are written to disk.

- ▶ Component: Queue destination and Topic destination
- ▶ Applies to specific messaging topologies: Yes
- ▶ Requires specific usage of JMS in application: Yes
- ▶ Location of setting: Varies - see text.

The configuration of whether a message should be persisted is configurable at three levels:

- ▶ Application level - in the application code

When using a `javax.jms.MessageProducer` it is possible to specify whether messages should be persistent. This is done using the `setDeliveryMode()` method which accepts values of either:

- `DeliveryMode.NON_PERSISTENT`
- `DeliveryMode.PERSISTENT`

When creating a message producer, if `setDeliveryMode` is not used then the default is `DeliveryMode.PERSISTENT`.

► Queue destination and topic destination settings

Within these objects it is possible to define the persistence. Possible values for persistence are:

- APPLICATION DEFINED
- PERSISTENT
- NON PERSISTENT
- QUEUE DEFINED (only on the WebSphere MQ JMS provider objects)

► Queue definition in queue manager

For WebSphere MQ it is possible to set the Default Persistence.

Even though you can configure the persistence at each level there are some rules that govern which setting will take precedence:

- Setting the value on the WebSphere MQ queue is only the default persistence setting. It is overridden by the setting on the JMS queue and topic destinations.
- Any value set in the application code will be overridden by the settings on the queue or topic destination unless they are configured as APPLICATION DEFINED, in which case the value in the application will be used.
- Defining PERSISTENT and NON PERSISTENT on the queue and topic definitions is the only way to accurately know from looking at the WebSphere Application Server configuration whether persistence will occur.
- If you are using WebSphere MQ (as opposed to the default messaging provider) then you can configure the queue or topic destination to take the default value specified on the queue. This is done using QUEUE DEFINED.

As more messages are stored on disk, a reduction in performance of message processing occurs. Persistence should only be used when the topology in use demands it. For example, if you decide to use one queue manager and have it made highly available using some HA software like HACMP, then you will need to have persistent messaging turned on to make this work.

Alternatively, if you have an architecture whereby messages are sent and received in a synchronous fashion, then it might be the case that the client code only waits for a certain amount of time before re-sending the message. In this instance the topology contains more than one active queue manager and persistent messages are not required since failures can be resent.

It is not straight forward to decide whether persistence is needed and for this reason it is much better that it not be part of the considerations for a developer. It is recommended that persistence is not specified by the application and is left to the definition of the queue or topic destination so that it is obvious to the administrator whether messages will be persisted.

Attention: When a queue or topic destination is defined the default is APPLICATION DEFINED. As the default for creating a new MessageProducer object in the code is to have messages persisted, the overall defaults for setting up a queue destination are that the message will be persisted. If you do not need messages to be persisted then you need to change the APPLICATION DEFINED to NON PERSISTENT on the queue or topic destination.

XA enabled resources

XA support extends transaction support as it allows WebSphere Application Server to coordinate a transaction over many back end resources, committing the results to those resources only when all the work in the transaction has been completed.

- ▶ Component: Connection factory
- ▶ Applies to specific messaging topologies: Yes
- ▶ Requires specific usage of JMS in application: No
- ▶ Location of setting:

Resources -> JMS Providers -> WebSphere MQ -> WebSphere MQ connection factories -> <ConnectionFactory_Name> -> XA enabled checkbox

Two-phase commit can only operate with XA enabled resources and this setting specifies this.

By default, XA support is enabled when creating a connection factory. With XA enabled more work has to be done to complete the transaction. It is an expensive overhead if there is no use of two-phase commit in your application and should be disabled in this case. Turning this off does not change behavior of a transaction, if XA enabled is set to false and the application tries to enlist more than one resource in the transaction, then an exception will be thrown.

If your application does need to use an XA enabled connection factory then you should establish whether it is needed for *all* interactions with the messaging system. If it is only needed for a small section of the application requests then it is recommended that two connection factory objects are created, one with XA

enabled and one without. This should then make the parts of the application that do not need XA run faster.

Client ID and Enable Clone support for publish/subscribe

The enable clone support setting is specifically for durable subscription based publish/subscribe messaging on the WebSphere products. Enable clone support permits listener ports defined with the same connection factory to share a durable subscription. This allows, for example, a MDB in an application deployed to a cluster to share a durable subscription such that each message is only sent to one MDB instance in the whole cluster.

- ▶ Component: Connection factory
- ▶ Applies to specific messaging topologies: Yes
- ▶ Requires specific usage of JMS in application: No
- ▶ Location of settings:

Resources -> JMS Providers -> WebSphere MQ -> WebSphere MQ connection factories -> <ConnectionFactory_Name> -> Enable clone support checkbox

When using publish/subscribe there are two ways in which client subscriptions can be handled, durable and non-durable. If a client subscribes to a topic using a non-durable subscription and then is terminated or loses network connectivity, it will not receive any future publications until it is started again. Any publications occurring while the client is unavailable will be lost.

Durable subscriptions mean that a clients subscription lives past the termination or loss of communication of that client. While the client is unavailable, the broker keeps a copy of any published messages requested in the clients durable subscription. When the client starts back up again the messages are then received by the client.

Important: As a durable subscription lives past the termination of the application client, the only way to remove the durable subscription is to deregister it from the broker. If you are using a durable subscription on an MDB then the durable subscription is only removed when the application is removed from WebSphere Application Server, and only then when that application server is restarted so that it can tell the broker of the subscription change.

There are different rules for how to create a durable and non-durable subscription, because of the persistence needed for a durable subscription.

A non-durable subscription works in a similar way to listening on a destination for a message to arrive. In the case of a message-driven bean, WebSphere Application Server uses a listener port to create the necessary JMS objects to point to the required topic. If any of those objects are closed or the link with the message broker is lost then the subscription is also lost. From the broker's perspective it recognizes a distinct subscription by the MessageConsumer that has been used. Each communication channel opened to the broker is a new subscription from a new client.

A durable subscription cannot be identified by the connection to the message broker as once it is closed or lost, so would be the subscription. Instead, the broker recognizes the subscription based on the ID of the durable subscription. This ID is based on a set of parameters which form a unique reference. If the client should stop, then when it is started again it will be recognizable to the broker by the fact that it has the same ID.

If a client, in particular an MDB with a durable subscription, wishes to subscribe, it needs to have a unique durable subscription ID. This ID is created using the:

- ▶ J2EE name of the MDB
- ▶ The client ID setting on the connection factory assigned to that MDB's listener port

Problems start to occur if the application is clustered. If there is only one installation of the application running, then this MDB's subscription remains unique for this broker. But if another copy of the application is started up pointing at the same connection factory then the subscription is no longer unique on the broker. The second MDB will not function as expected.

For this reason, when clustering an application you need to carefully choose how many connection factories serve your MDBs and the settings on those factories.

Two different types of behavior with a durable subscription and MDBs are possible:

- ▶ The application is installed in a cluster. All the MDB listener ports are configured to point at one connection factory, which is defined at the lowest scope that allows all your application servers to view it. This will be either cell or node depending on your topology. This connection factory has **Enable clone support** checked and a **Client ID** specified.

When all the servers in your cluster are started, the MDBs and their listener ports start successfully. When a message is published only one of the MDBs will get the message, so only one of the cluster members will receive the information within the message.

This is because the broker only sees one subscription as the client ID on the connection factory and the name of the MDB are the same for each application server. It is at this point that the Enable clone support setting needs to be used. The JMS specification states that there can only be a single subscriber for a particular durable subscription at any one time. Enabling clone support allows all the MDBs to successfully start even though they all have the same client ID for the durable subscription. Without this setting turned on, only one of the MDBs will work properly. With Enable clone support turned on, the JMS provider will then pass a publication to any one of the MDBs.

Whichever of the listener port sessions is available first will receive the message. This is like one grouped line of customers waiting to be served by a number of cashiers, where the line of customers is the topic publication messages and the cashiers are the listener ports. There are four cashiers working, each dealing with a customer, when one of them finishes their work the next customer in the line goes to that cashier. If a line never forms then the cashiers will not be busy and so the next customer to arrive could go to any of them. This method of distributing the workload means there is no algorithm that can be applied to understand which cluster member will get any given message.

This behavior occurs because the broker sees one subscription. When a message is published that matches this subscription, the MQ JMS classes take the message and then pass it to whichever of the listener ports is available first to process the message. This behavior occurs because Enable clone support has been specified, it is telling the MQ JMS provider to allow multiple MDBs (one in each clone), but only use one at a time.

Note: This scenario means that all MDBs use the same communication mechanism to reach the message broker.

- The application is installed in a cluster. Each MDB listener port that is defined has its own connection factory, which is defined at the server scope. Each connection factory does not need clone support checked, but does need a client ID specified. The client ID field must be a unique value among all the connection factories defined to use the same broker.

When all the servers in your cluster are started, the MDBs and their listener ports will start successfully. When a message is published, all of the MDBs will get the message, so all of the cluster members will receive a copy of the message. This behavior occurs because the broker sees multiple subscriptions, one for each application server.

Conclusion: If you are using durable subscriptions in a clustered environment then it is essential that the correct behavior is created through the configuration of your MQ JMS components.

13.3.3 The listener service and listener ports

11.4.1, “Basic workload patterns” on page 627 and “Component relationships when using MDBs” on page 705 have already discussed the way in which workload arrives via the messaging system and the relationships between each of the settings for the message listener service. This section provides details about specific settings that can help to optimize the message listener service and MDBs.

Message listener port maximum sessions

When using MDBs within an application it is recommended that advantage is taken of the ability to process messages simultaneously. This is done by increasing the maximum number of sessions on any listener ports that are configured. By default, Maximum sessions is set to 1.

- ▶ Component: Message listener
- ▶ Applies to specific messaging topologies: No - as long as MDBs are used
- ▶ Requires specific usage of JMS in application: Yes
- ▶ Location of setting:

Servers -> Application servers -> <AppServer_Name> -> Messaging -> Message Listener Service -> Listener Ports -> <ListenerPort_Name> -> Maximum sessions field

Deciding on the value for this field involves performance testing as there are a number of factors that determine its optimal value. There are two factors to consider:

- ▶ Time to complete onMessage method of MDB
The time that it takes to process a message off the message queue can be used to determine how long resources are in use. When a message is passed to an onMessage method of an MDB there is one listener port session, one session and one listener service thread in use. All of these resources are locked while that message is being processed. The shorter the service time of the onMessage method, the less time the resources are locked and are free to process another message.
The average time it takes for the onMessage to complete on an MDB can be found using the Tivoli Performance Viewer. See “Monitoring performance with Tivoli Performance Viewer” on page 765. To be accurate, this average time

must be measured when a realistic application workload is being tested in a realistic production environment.

► Peak workload arrival rate

The advantage of having multiple sessions is that up to x messages from the destination may be processed simultaneously, where x is the total number of listener port sessions. The peak workload will affect the value for the number of maximum sessions.

In a system where the time between each message arriving is more than the time it takes to process one message, adding more message listener sessions is not going to give any performance improvement as there will be no messages waiting on the queue.

However, if the messages cannot be processed faster than the arrival rate then there is scope for increasing the maximum sessions value to prevent a backlog of messages occurring.

Important: The total processing time for a message includes not only how long the message takes to be processed once it has been delivered to the MDB but also the time it takes waiting to be processed. Only the time that it spends waiting can be optimized by changing the maximum number of sessions.

Understanding the peak and average workload arrival rates are essential in configuring the optimal value for maximum sessions for your application running in your environment.

If possible, come up with an estimate of the maximum number of messages that will arrive on the message queue over a given amount of time that makes sense in your application (second, minute, etc.).

As with all performance tuning, the correct balance between efficient usage of system resources and maximizing throughput needs to be reached. If the workload does arrive faster than messages can be processed then increasing the maximum number of sessions on a listener port will only increase performance if there is spare resource available on the physical server to perform the simultaneous processing of messages.

It is always worth monitoring the number of sessions in use on a listener port. If a test shows that setting the Maximum sessions to 10 did not perform any faster than Maximum sessions set to 5, then use Tivoli Performance Viewer to check that all 10 sessions were actually in use. It might be the case that not enough workload is arriving to push the system to use all 10 sessions, in which case look at the frequency that the message sender is capable of placing messages on the queue.

There is, however, a limit in gains when increasing the number of maximum sessions that does not have to do with hardware restrictions. This is because there is only one queue reader thread per queue per JVM. This is the thread that is farming out work to the sessions. So as you increase maximum sessions, the queue reader thread receives a higher load.

The messaging system topology and WebSphere Application Server topology make a difference on how the peak workload arrival rate is measured and how long a message takes to process. The topology used in “Clustered Trade 6 with WebSphere MQ and WebSphere Business Integration Event Broker” on page 732 has two application servers pointing at the same message queue for redundancy purposes. This means that the message queue is not solely accessed by one message listener service.

While the time to complete the message and the workload arrival rate have a significant impact on how many sessions should listen for messages, they are not the only settings, and are in themselves affected by a great number of other factors. The only reliable way to determine the appropriate number of maximum sessions on a listener port is through thorough performance testing and tuning.

Tip: Having trouble getting the Maximum sessions value to increase above 1 in tests? Your messages are probably being processed faster than the next one is arriving. One way to simulate the peak workload arriving at the message queue is to stop the message listener port in the Administrative Console, but continue to forward messages to the message queue. This will fill up the queue with messages. Once the number of messages on the queue has reached the desired amount, start the message listener port up again.

One final point to reemphasize is that increasing the maximum number of sessions on a listener port requires increasing other values too, check “Component relationships when using MDBs” on page 705 to find out what else needs to be tuned.

Maximum messages

This setting controls the maximum number of messages that the listener can process in one session.

- ▶ Component: Message listener
- ▶ Applies to specific messaging topologies: No
- ▶ Requires specific usage of JMS in application: Yes
- ▶ Location of setting:

Servers -> Application servers -> <AppServer_Name> -> Messaging -> Message Listener Service -> Listener Ports -> <ListenerPort_Name> -> Maximum messages field

The MQ JMS based provider handles the starting of the MDBs transaction differently than other JMS providers.

When the MQ JMS provider receives one or more messages it retrieves a listener port session, gets the JMS session from it and passes the message to that session. The JMS provider then calls start on the listener port session which passes the MDB reference to the JMS session, spins off a new message listener thread and calls run on the JMS session. The MQ JMS provider then delivers the message to the onMessage method of the MDB.

Changing the value of Maximum messages increases the number of messages delivered per transaction. However, this setting might not have an obvious impact on performance, and it changes the way in which those messages are handled:

- ▶ If one message in the batch fails processing with an exception, the entire batch of messages is put back on the queue for processing.
- ▶ Any resource lock held by any of the interactions for the individual messages are held for the duration of the entire batch.
- ▶ Depending on the amount of processing that messages need, and if XA transactions are being used, setting a value greater than 1 could cause the transaction to time out.

The performance improvement will not be obvious when changing this setting as messages will not necessarily be processed faster. The correct ratio between Maximum messages and Maximum sessions would need to be reached.

For example, if there are 10 messages on a queue and the Maximum sessions is set to 5, Maximum messages is set to 1, then 5 messages at a time will be processed (assuming enough physical resources). If Maximum messages is set to 5 and Maximum sessions remains at 5, then 5 messages will go to the first session and 5 messages will go to the second session. This means that only 2 messages will be processed at the same time.

The workload will impact how effective changing the maximum messages setting will be. In our example, if the number of messages is increased to 50 or more then you should start to see some benefit of the reduced overhead of creating transactions.

Message listener service thread pool

Each listener port defined on an application server's message listener service will have a maximum number of sessions that it can process at the same time.

Each session requires one message listener thread to be able to operate. The listener service thread pool must have enough threads available at any one time to allow all the listener ports sessions defined under it to start. This means that if there are four listener ports defined, and between them the maximum sessions settings add up to 20, then the maximum size for the thread pool should be no less than 20.

- ▶ Component: Message listener
- ▶ Applies to specific messaging topologies: No - as long as using MDBs
- ▶ Requires specific usage of JMS in application: No
- ▶ Location of setting:

Servers -> Application servers -> <AppServer_Name> -> Messaging -> Message Listener Service -> Thread Pool

The minimum size and thread inactivity time-out should be used together to balance how much JVM resource and heap is taken up by the inactive threads versus the performance impact of having to start up new threads. Depending on your priorities this could mean setting the minimum size to the same as the maximum size to remove the overhead of creating threads. It could mean setting it to 1 and using a large inactivity time-out so that the thread pool does not use system resources until threads are needed, but during a busy time the overhead of the threads being created only impacts the first messages to arrive.

Underlying MQ JMS connection pooling

JMS connections and sessions are pooled at the Java layer by WebSphere Application Server. At a lower level the MQ JMS classes also pool the physical connections to the MQ queue manager. This pooling is done automatically by the MQ JMS classes when accessing WebSphere MQ. It is possible to change two of the parameters that control the cleanup of idle connections in this pool should you need to.

There is no association between the MQ JMS connection pool and the connection factory connection or session pool. This lower level connection pool is used by the MQ JMS classes when needed.

MQJMS.POOLING.TIMEOUT

By default, the MQ JMS pooling will maintain each established idle connection to the queue manager for five minutes. Change this value to specify a different lifetime for an idle connection. The value should be entered in milliseconds.

MQJMS.POOLING.THRESHOLD

The MQ JMS connection pooling is also controlled by the number of idle connections in it. If the number of idle/unused connections reaches the default of

10 connections then any other connection becoming idle will be destroyed. This is basically the minimum size for the pool. Changing this value will allow you to control how many established connections can survive for the time-out setting above.

Use these two settings to control how many low level connections are available in the pool and how long they will survive if they continue to not be used. Follow these steps to use these two values.

1. Go to **Servers -> Application servers -> <AppServer_Name> -> Messaging -> Message Listener Service -> Custom Properties**
2. Click the **New** button.
3. For name specify either:
 - MQJMS.POOLING.THRESHOLD
 - MQJMS.POOLING.TIMEOUT
4. Enter the value you wish to use, and an optional description. Remember the time-out is in milliseconds.
5. Click **OK** and save your configuration. The settings will come into operation the next time the application server is restarted.

Use a separate connection factory for MDBs

Each MDB's listener port is associated with a connection factory. The connection factory objects hold the connection and session pools for accessing the messaging system. When configuring the sizes of these pools there always have to be enough sessions and connections available to allow all parts of the application to access the messaging system at peak load. The more variants there are in working out this peak load the harder it is to get the correct figures for the size of the pools.

The number of connections and sessions that are needed by each listener port on an application server can be calculated. The number of connections and sessions used by the Web and EJB container cannot be calculated in the same way as it is reliant on the type and spread of workload, which can vary considerably. Only a percentage of requests that arrive at the Web or EJB container will need to use JMS, unlike the listener port where 100% of requests (messages) need to use JMS.

Creating a separate connection factory definition that will only be used by MDBs means that:

- ▶ There will always be enough resources available in the connection and session pool to run the listener ports that push work to the MDBs. This is regardless of whether the Web or EJB container are experiencing heavy load.
- ▶ There is one less variable to consider when setting up the connection and session pools for the Web and EJB traffic.
- ▶ When monitoring performance it is obvious which connections and sessions are in use by listener ports. In Tivoli Performance Viewer, objects are shown by the name of the connection factory they use.

For each connection factory associated with a listener port:

- ▶ Set the minimum connection pool size to 1 and the maximum connection pool size to 2. The listener port will only need one connection; setting the maximum to 2 is just a precaution.

Assuming the MDB application code is using the same connection factory as the listener port, if the MDBs themselves perform or call code that needs to put a message on a destination using the same connection, do not use these settings. The details of your application determine how many more connections are needed to handle the sending of the message from within the MDB's transaction.

- ▶ Set the maximum session pool size to the same value as Maximum sessions in the Listener port configuration.

Once again, if the MDB or code it calls puts a message on a message destination using the same connection factory, then more sessions might be needed than this.

Failure behavior settings

There are a number of settings that define how the listener service and listener port operate in the event of a failure. Optimizing these settings will have an impact on performance in the event of a failure. Go to “JMS Listener port failure behavior” on page 726 for information.

13.3.4 Setting up the connection factory pools

When using JMS, the connection and session pools are the link between the application and the messaging system. There need to be enough connections and sessions available in each connection factory to satisfy the needs of the peak workload.

- ▶ Component: Connection factory

- Location of settings:

Resources -> JMS Providers -> WebSphere MQ -> WebSphere MQ connection factories -> <ConnectionFactory_Name> -> Connection pool
(from the Additional Properties pane)

Each connection factory will have a different anticipated workload, so it will need configuring to match that workload. Finding the optimum value for the minimum and maximum connections and sessions in the pools can only be achieved through performance testing and tuning.

However, it is possible to help in finding the correct starting place for maximum pool sizes to use in the performance testing. It is how the connection factory is going to be used that will help in deciding what the size the pools should be. Consider the areas listed below when setting up the connection and session pools:

- What is the intended usage of the connection factory?
 - MDB Listener port
 - XA enabled resources
 - Generic JMS
- Are there going to be different connection factory objects for each of the above?

If so then adjust the size of the pools accordingly (minimum and maximum connections for each pool).
- For generic JMS, what is the relationship between connections and sessions?

See “MQ JMS component relationships” on page 705. Remember that each connection has its own pool of sessions, but each session pool is globally configured on the connection factory.
- How many Web and EJB threads are there defined in the application server?

The Web and EJB containers maximum thread pool sizes act as the controlling mechanism for how many threads could be trying to use the connection factory. Whenever the number of threads is increased or decreased, consider changing the connection and session pool sizes as well. See “MQ JMS component relationships” on page 705 for more information.

13.3.5 More information

For more information about configuration settings that affect JMS performance, including some WebSphere MQ specific settings, refer to the WebSphere Application Server 6 InfoCenter:

<http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp>

Search for “Tuning service integration messaging” in the InfoCenter.

Also take a look at the supportpacs available about JMS performance with WebSphere MQ:

<http://www.ibm.com/software/integration/support/supportpacs/product.html>

13.4 JMS Listener port failure behavior

Within WebSphere Application Server, the listener port is the component on the message listener service that links a message-driven bean to a connection factory and destination, and therefore to the underlying messaging system.

There are two scenarios that need to be catered for when configuring the listener port and message listener service.

13.4.1 Failure in the listener port

If a listener port should fail to start, or lose its connections to the MQ queue manager then it is possible to make it retry. There are two settings at the message listener service that govern how many times restarting a listener port should be attempted and how long the interval should be in between.

These two settings are:

- **MAX.RECOVERY.RETRIES**

Use this to specify the number of attempts to start the listener port. The default for this setting is 5 attempts.

- **RECOVERY.RETRY.INTERVAL**

This setting sets the time interval between the retries. The default for this value is 60 seconds.

If these settings are not changed then the listener port will attempt to start once every 60 seconds over 5 minutes. If the listener port has been unable to start in this time then it will not be started, unless done manually.

Change these values to provide an adequate amount of retries so that your application can recover. For example, if you are using hardware or operating system level clustering to provide fail over, then these settings need to be configured to keep retrying for the period it typically takes for fail over to occur. In this way the failure of the queue manager will be masked as a blip in throughput rather than the listener port stopping completely.

Follow these steps to use these two values:

1. Go to **Servers -> Application servers -> <AppServer_Name> -> Messaging -> Message Listener Service -> Custom Properties**
2. Click the **New** button.
3. For name specify either:
 - MAX.RECOVERY.RETRIES
 - RECOVERY.RETRY.INTERVAL
4. Enter the value you wish to use, and an optional description. Remember the retry interval is in seconds.
5. Click **OK** and save your configuration. Once completed it will look similar to Figure 13-6 on page 727.

The settings come into operation the next time the application server is restarted.

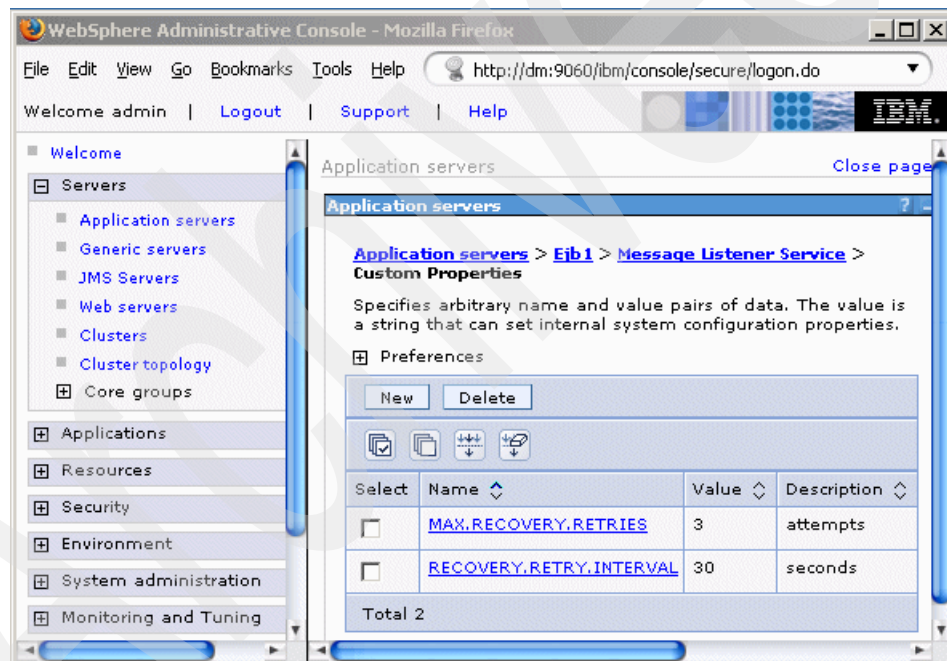


Figure 13-6 The listener port retry settings

Attention: This behavior is slightly different for connections to a message broker. When the JMS provider code attempts a connection to the message broker for publish/subscribe messaging, if there is a problem getting a response from the broker then it will wait for three minutes before timing out. This means that if a listener port is trying to start or restart there will be an additional timeout value added on between retries to the retry interval. This message broker connection timeout is not configurable from within WebSphere Application Server.

13.4.2 Failure to process a message

The listener port will retrieve the messages from the message queue and pass them to the MDB. If there is an error in processing a message then the MDB will rollback and the message listener port will return that message to the queue. Of course, once back on the queue this message will be picked up again by the listener port and a never ending cycle could occur.

There are some settings that can stop this occurring. On each listener port it is possible to configure the *maximum retries* setting. By default this setting is set to 0, which means if the processing of one message fails then the listener port will stop. This is good for stopping the message from getting stuck in a never ending loop but it also means that all the other messages waiting on the queue will not get picked up.

It is possible to change the maximum retries to a higher number and then this might give the message a chance to succeed. For example, if the message was being used to update a database but access to the database timed out then retrying might eventually work.

Even changing the maximum retries to a higher number might still result in the listener port stopping.

There is a way to stop this behavior occurring and that is through the use of backout queues on WebSphere MQ. Within the configuration of a queue in WebSphere MQ, it is possible to set a backout queue and backout threshold. The backout threshold works in the same way as the listener ports maximum retries. It counts the number of attempts at delivering a message. Upon reaching the backout threshold number of retries, the message is taken from the queue and placed on the queue specified in the backout queue.

These settings for the queue can be set using the MQ Explorer GUI as shown in Figure 13-7 or using the command line.

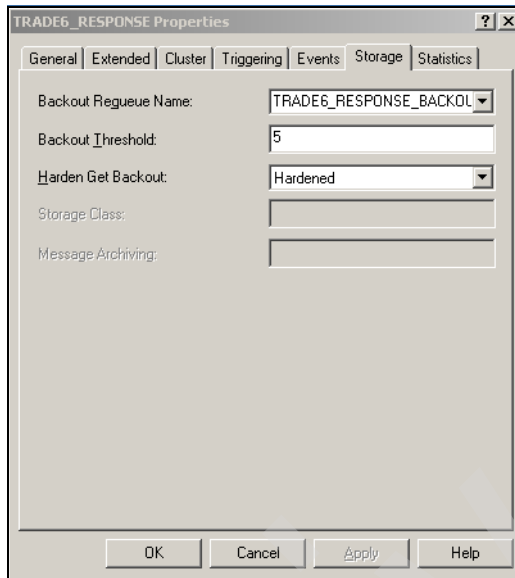


Figure 13-7 The backout queue settings for a message queue.

Make sure to set the Harden Get Backout field to Hardened so that an accurate count of message deliveries is kept.

This now means that the listener port stopping can be avoided by making the backout queue threshold lower than maximum retries. If the maximum retries is set to 3 on the listener port and the backout threshold set to 2, then the message will only ever get delivered twice and so not trigger the setting on the listener port. The message will be placed in the backout queue and perhaps another part of the application can be used to handle the backout queue.

The failed message will be handled and the message listener can continue to run. However, if for example, the MDB is trying to use a database that is down, then all of the messages are routed to the dead letter queue.

13.5 Example JMS topologies and scenarios

A lot of the settings that have been discussed have topology specific benefits. Functionality and configuration are a lot more significant when designing a messaging and application server topology for more than traditional Web and EJB based applications. In these cases it is the performance and scalability requirements that often force the choice of topology.

When using JMS, there is not one topology at the top of the food chain that will suit all situations. Using point-to-point, publish/subscribe, MDBs, different messaging products, and combinations of these will all drive a need to have a specific topology to match a specific application and its non-functional requirements.

The choice of topology will depend on the correct balance of non-functional requirements of performance, security, scalability, maintenance, reliability, availability and cost, as well as providing a topology that can give the required function.

The example application that has been used throughout this book is Trade 6. Trade 6 has specific requirements for its use of JMS. Trade 6 uses JMS as an asynchronous communication mechanism between two parts of the application. This means that for Trade 6 there is no need to communicate with a back-end system and so a lot of the more complicated topologies do not suit it.

However, this chapter is supposed to be helping you understand WebSphere MQ, so for the purposes of this section, we describe a scenario using Trade 6 with requirements for functionality and other criteria. The front end sections of the topology (Web traffic) are not discussed but it is possible to overlay the sample topology in Chapter 8, “Implementing the sample topology” on page 387.

This section is intended to help you use some of the configuration options described in this chapter in working examples. It should also help you begin to understand the thought process in deciding on the best topology for your application. It will not describe all possible topologies, and there are many other topologies. To find out more on choosing a topology take a look at the document *JMS Topologies and Configurations with WebSphere Application Server and WebSphere Studio Version 5* available at:

http://www.ibm.com/developerworks/websphere/library/techarticles/0310_barci_a/barcia.html

Note: This document is currently being updated for WebSphere V6 and will become available on developerWorks in the near future.

13.5.1 What does Trade 6 use JMS for?

Before continuing to the examples it is important that the functional requirements for JMS in Trade 6 be clear.

Trade 6 integrates both point-to-point and publish/subscribe MDBs.

Point-to-point

The queue based TradeBrokerMDB asynchronously processes stock purchase and sell requests. In the Trade 6 configuration, if the Order Process mode is set to Asynchronous_1-Phase or Asynchronous_2-Phase then instead of an order for some stock being processed immediately, it is handled asynchronously. Upon buying some stock an order is opened in the Trade database and a message sent to a queue. The application then immediately returns to the client. In the background the message is picked up by the TradeBrokerMDB and processed, updating the records in the database to set the order to completed. The next time the client returns to the Web site, they receive a notification that the order completed successfully.

This is using JMS for asynchronous communication within the application. Some important points to note on the influence the functionality has on the topology are:

- ▶ The topology required for this part of Trade 6 does not require communication with any back-end systems.
- ▶ Any cluster member can process the order messages as they update the database. This means that if the client is on TradeServer1 and the message is processed by TradeServer2, the client still receives the notification. This is asynchronous communication with no need for affinity.
- ▶ Having one or more WebSphere MQ queue managers will not affect the functionality of the application, nor will using messaging clients.

Publish/Subscribe

TradeStreamerMDB subscribes to a message topic called TradeStreamerTopic. Quote data for individual stocks is published to this topic as prices change, providing a real-time streaming quote feature in Trade 6. The prices change every time some stock is bought or sold.

Some important points to note on the influence the functionality has on the topology are:

- ▶ If the Trade 6 application is clustered, then each application server needs to get the publications on price changes. If this does not occur then the prices will not be kept in synchronization across the cluster. This means having only one message broker doing the work (or a cluster of message brokers).
- ▶ By default, the MDB is using non-durable subscriptions. This avoids any restrictions imposed from using durable subscriptions. See “Client ID and Enable Clone support for publish/subscribe” on page 715 for more information.

Now that we understand how Trade 6 needs to use JMS we move on to apply some of the guidelines described in this chapter and create some topologies.

13.5.2 Clustered Trade 6 with WebSphere MQ and WebSphere Business Integration Event Broker

The main purpose of this example is to show how WebSphere Application Server, WebSphere MQ and WebSphere Business Integration Event Broker can be used to create a scalable infrastructure. This is to be done through the use of both WebSphere Application Server clustering and WebSphere MQ clustering. This section is aimed at WebSphere Application Server users who have had minimal exposure to WebSphere MQ and WebSphere Business Integration Event Broker.

To make this scenario more realistic some requirements have been set:

- ▶ A new functional requirement has been requested. When stock is bought or sold it needs to be processed by a back-end system first. A response message will then be generated by the back-end and the Trade 6 database will be updated as normal.

The reason this new requirement has been specified is because WebSphere MQ clustering only workload manages the sending of messages to remote clustered queues. If the back-end processing is not needed then WebSphere MQ clustering does not aid asynchronous communication between two parts of the same application.

This is explained further in “Initial WebSphere MQ configuration” on page 736.

- ▶ Performance, reliability, and availability are of equal importance. The system must perform well, have minimal outages, and should an outage occur it needs to be able to recover.
- ▶ The application must link into an existing messaging system which is based on WebSphere MQ.

There would normally be other factors that influence the choice of topology, but this should be enough to justify the following topology.

The chosen WebSphere MQ clustering topology

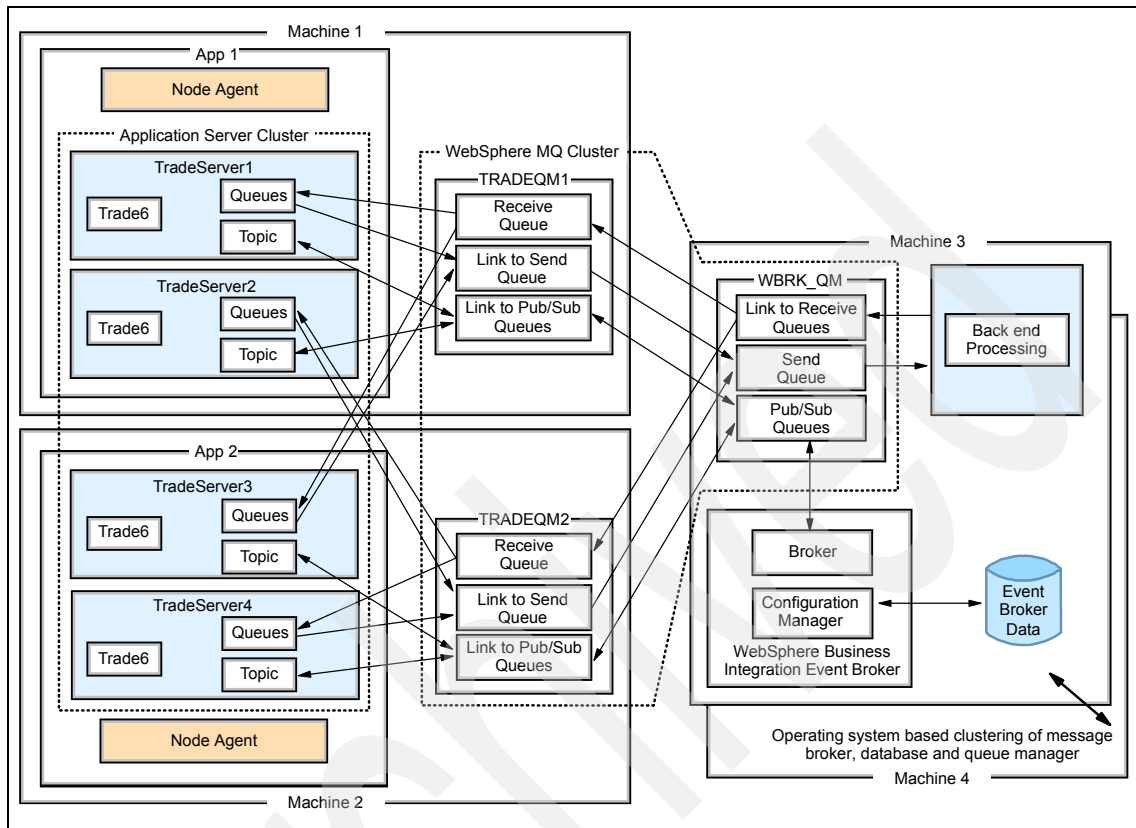


Figure 13-8 Example topology for WebSphere MQ and Event Broker for Trade 6 scenario

This topology has been chosen because:

- It provides workload management.

There are four servers in the application server cluster that have workload distributed to them, both from the Web facing side and from the messaging side. The queue managers also have their workload managed when requests are returned from the back end system. Finally, although not specified on the diagram, there is potential for the broker to be workload-managed as well.

- It has been designed for high availability and reliability of WebSphere Application Server.

Four application servers are used to cover the chance of an application server failing. In this topology, messages arriving from the back end are workload-managed between any available queue manager. If TRADEQM1 and TRADEQM2 are both up then response messages will continue to arrive

at both. If TradeServer1 was stopped and TradeServer2 did not exist then messages would continue to arrive at TRADEQM1 but would not get processed.

This is overcome in this example by having one application server on each node pointing at the remote queue manager. Should TradeServer1 stop, any arriving messages will be picked up by the MDB running on TradeServer2.

There is still the issue that should TRADEQM1 or TRADEQM2 stop then two of the four application servers will not be able to process order requests that will still be coming in from the front end. This failure in sending of messages can be avoided by changing the Trade 6 code to have two connection factories to try, a primary and a backup. Upon sending a message if the first connection factory fails to respond then the second could be used. This code change will not be made for this example.

- It has been designed for high availability and reliability of WebSphere MQ.

In this topology, the WebSphere MQ infrastructure uses clustering to provide failover and workload management. This prevents the queue managers becoming a single point of failure for messages arriving from the back end system. See “Initial WebSphere MQ configuration” on page 736.

- It has been designed for high availability and reliability for WebSphere Business Integration Event Broker.

WebSphere Business Integration Event Broker and its supporting software of a WebSphere MQ queue manager and DB2, are made highly available through the use of operating system level clustering software, for example HACMP for AIX. The setup of these products on various types of clustering software is out of the scope of this book. Please refer to the redbook *WebSphere Application Server Network Deployment V6: High availability solutions*, SG24-6688 for more information about this topic.

In this topology there is a reliance on the queue managers TRADEQM1 and TRADEQM2 being available for pub/sub messages to be delivered. The application servers use the two local queue managers as routing mechanisms to reach the message broker.

- Through use of durable subscriptions, published messages are delivered regardless of the state of the client at publish time.

If the MDB in Trade 6 remains using a non-durable subscription as it is currently set, then the subscription to the broker is based on the connection the MDB listener has acquired to its local queue manager. If that connection is broken then the MDB loses its subscription. As this system needs to be reliable and recoverable from failure, a durable subscription needs to be used and the infrastructure setup accordingly. A durable subscription will live past any failure to communicate with the client, storing undeliverable messages until the client comes back online.

- ▶ It provides a topology that fits the functional requirements and allows communication with other WebSphere MQ queue managers outside of the WebSphere Application Server cell.

Important: This is not the best topology for all applications in all cases.

The back-end application has been included into the additional materials repository of this redbook. Refer to Appendix B, “Additional material” on page 1037 for download instructions. It is a very simple EAR file called Trade6Redirector.ear, that contains one MDB. This MDB listens for arriving messages on one queue, picks up the messages and puts them back on the response queue. Basic instructions for installing it are included in the EAR file package.

The following steps for WebSphere MQ and WebSphere Business Integration Event Broker have been designed to be the easiest method of getting a skilled WebSphere Application Server architect or administrator up and running. However, they might not be the most efficient method to complete the tasks needed under every networking and hardware environment.

Software

The following software was used to perform this example. The steps in this example are for setting this up on a Microsoft Windows platform. It is assumed that before carrying on with these steps, the following software is installed:

Machine 1

- ▶ IBM WebSphere Application Server Network Deployment V6
- ▶ IBM WebSphere MQ 5.3 + fixpack 9
- ▶ IBM DB2 UDB Client version 8.2 (optional)

Machine 2

- ▶ IBM WebSphere Application Server Network Deployment V6
- ▶ IBM WebSphere MQ 5.3 + fixpack 9
- ▶ IBM DB2 UDB Client version 8 (optional)

Machine 3

- ▶ IBM DB2 UDB version 8.2
- ▶ IBM WebSphere MQ 5.3 + fixpack 9
- ▶ IBM WebSphere Business Integration Event Broker 5.0 + fixpack 4
- ▶ WebSphere Application Server - Express V6 or Network Deployment. This is to run the back end application.

When installing DB2, WebSphere MQ and WebSphere Business Integration Event Broker accepting the defaults should be enough for this example to work. Where any of the products require fixpacks it is recommended that the fixpacks be applied before starting up the products.

Important: When installing WebSphere MQ, be sure the Java Messaging component is selected for installation as shown in Figure 13-9. You may need to select **Custom** instead of a typical installation.

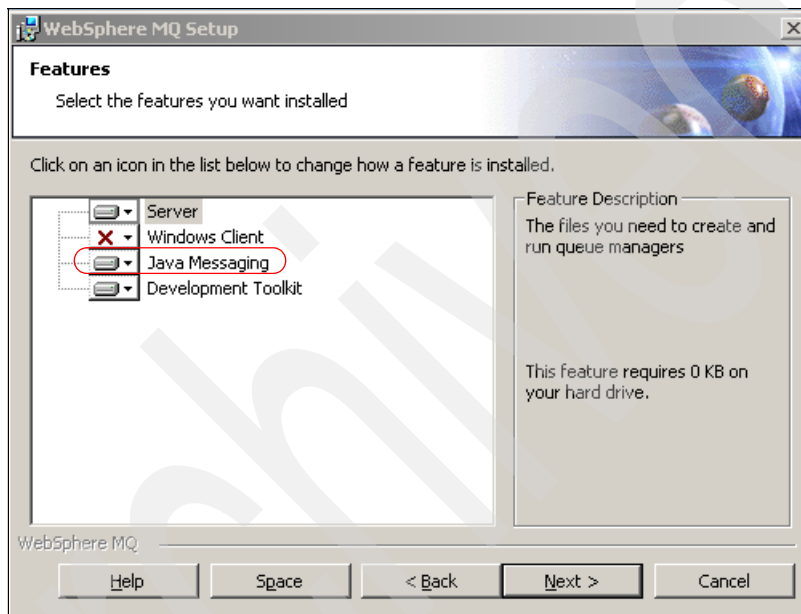


Figure 13-9 Selecting Java Messaging during WebSphere MQ install wizard

Machin 4

Is not covered in the setup.

It is assumed that WebSphere Application Server nodes app1 and app2 are federated into the cell.

Initial WebSphere MQ configuration

In this example the WebSphere MQ configuration underpins both how WebSphere Application Server and WebSphere Business Integration Event Broker operate. WebSphere MQ clustering facilities are used for workload management of messages, fail over and for ease of system administration.

A cluster is a group of queue managers set up in such a way that the queue managers can communicate directly with one another without the need for complex configuration. In addition to the WebSphere MQ manuals there is some good information about configuring WebSphere MQ clusters in chapters 8 and 12 of the redbook *Patterns: Self-Service Application Solutions Using WebSphere V5.0*, SG24-6591.

Figure 13-10 shows the queues that need to be configured for this example to work.

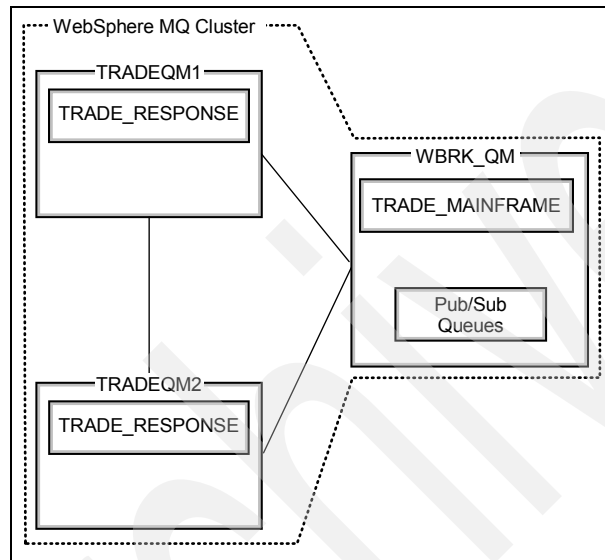


Figure 13-10 The queues in the WebSphere MQ cluster

The way in which MQ clustering works is to first join a number of queue managers together into a cluster (using commands or the GUI). This now means that each queue manager knows about any queue that has been configured as clustered. For example, WBRK_QM has a queue called TRADE_MAINFRAME defined under it and set as shared in the cluster. From TRADEQM1's perspective it can now see that there is a queue in the cluster called TRADE_MAINFRAME and it lives on WBRK_QM. Any client connecting to TRADEQM1 can simply specify that it wishes to place a message on TRADE_MAINFRAME and the WebSphere MQ cluster does the work of getting the message to WBRK_QM.

This is also how workload management occurs. WebSphere MQ clustering will only workload manage messages when there is more than one remote destination for the message. If a client connects to WBRK_QM to place a message on TRADE_RESPONSE then WBRK_QM can see two TRADE_RESPONSE queues, one on TRADEQM1 and one on TRADEQM2.

The workload management of WebSphere MQ will decide which queue to use. Should one of the queue managers not be available then the message will not be routed there, providing fail over too.

However, if a client connects to TRADEQM1 to place a message on TRADE_RESPONSE then that message will not get workload-managed and will go to the local queue on TRADEQM1 even though two queues with the name TRADE_RESPONSE are visible. (An exception to this is if the client specifies a different queue manager for the location of the queue, but this will still not be workload-managed).

Also, once a message is delivered to a queue manager that is the end of its journey until a client picks it up, even if a failure results in that queue manager. In a cluster there is no central repository of messages, a message either exists on the sending queue manager or the receiving queue manager.

Hopefully from this explanation you can now see why it was necessary to include the back-end process. Without the back-end process in this example no workload management or fail over of point-to-point messages would occur and so using WebSphere MQ clustering would not be valid for this application. If the only queue managers that existed were TRADEQM1 and TRADEQM2 then there is no need for WebSphere MQ clustering beyond configuration management.

Create queue manager on machine 1

Follow these steps to setup WebSphere MQ for use in this example:

1. Open the **WebSphere MQ Explorer**.
2. Right-click the **Queue Managers folder** in the IBM WebSphere MQ Explorer and select **New -> Queue Manager** from the pop-up menu.
3. In Step 1 of the wizard, shown in Figure 13-11 on page 739, set the Queue Manager name to **TRADEQM1** and make sure to specify the Dead Letter Queue of **SYSTEM.DEAD.LETTER.QUEUE**.

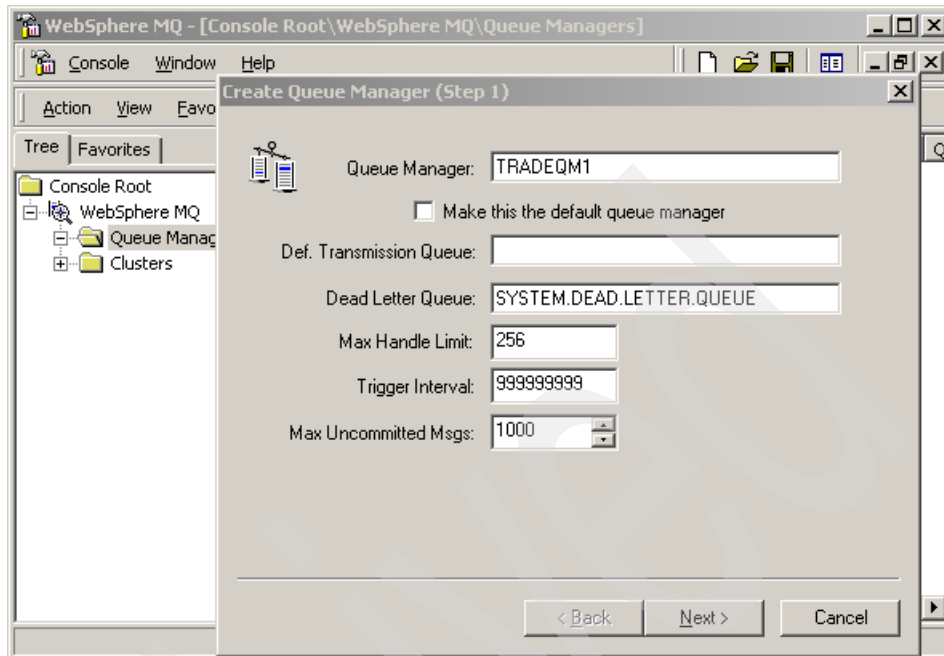


Figure 13-11 Create a new queue manager in WebSphere MQ

4. Click **Next**. Accept the defaults and click **Next** again.
5. On page three of the wizard make sure to check the box next to **Create Server connection Channel to allow remote administration of the queue manager over TCP/IP**. Click **Next**.
6. On page four of the wizard make sure to enter an unused **Port Number** for the queue managers listener. When creating this example, the Port number 1415 was used. Click **Finish**.
7. The queue manager will be created and started.
8. When using pub/sub and JMS there are a number of extra queues that are needed for it to work. To define these queues first open a Windows command prompt.
9. Go to the directory <MQ_Install_root>\bin. For example to c:\Program Files\IBM\WebSphere MQ\bin.
10. Run the command `runmqsc TRADEQM1 < MQJMS_PSQ.mqsc`
This creates all necessary JMS queues needed for publish/subscribe.

Note: In some versions of MQ, the bin directory is under <MQ_Install_root>\Java\bin. Also, if the MQJMS_PSQ.mqsc file is missing, you may need to search for it on your CD or install image and copy it. For our install image, it was located in \MSI\Program Files\IBM\WebSphere MQ\Java\bin.

Create queue manager on machine 2

Repeat the steps from “Create queue manager on machine 1” on page 738 for machine 2, changing the queue manager name to **TRADEQM2** and remembering to choose a **unique port** for the listener.

Create cluster

To make the steps simpler the creation of the queue manager on machine 3 will be handled later by the setup of WebSphere Business Integration Event Broker. So the next step is to configure the cluster.

1. On machine 1, open the WebSphere MQ Explorer.
2. Right-click the **Clusters folder** and select **New -> Cluster**.
3. This brings up a wizard to take you through the steps for configuring the cluster. Click **Next** when you have read the first page.
4. Enter **TRADE6** as the cluster name. Click **Next**.
5. On the third panel select the local queue manager **TRADEQM1** from the options. Click **Next**.
6. For the secondary repository queue manager setup the fields as shown in Figure 13-12 on page 741. Select **Remote**, enter the queue manager name **TRADEQM2** and the **host name and port** of the queue manager. Click **Next**.

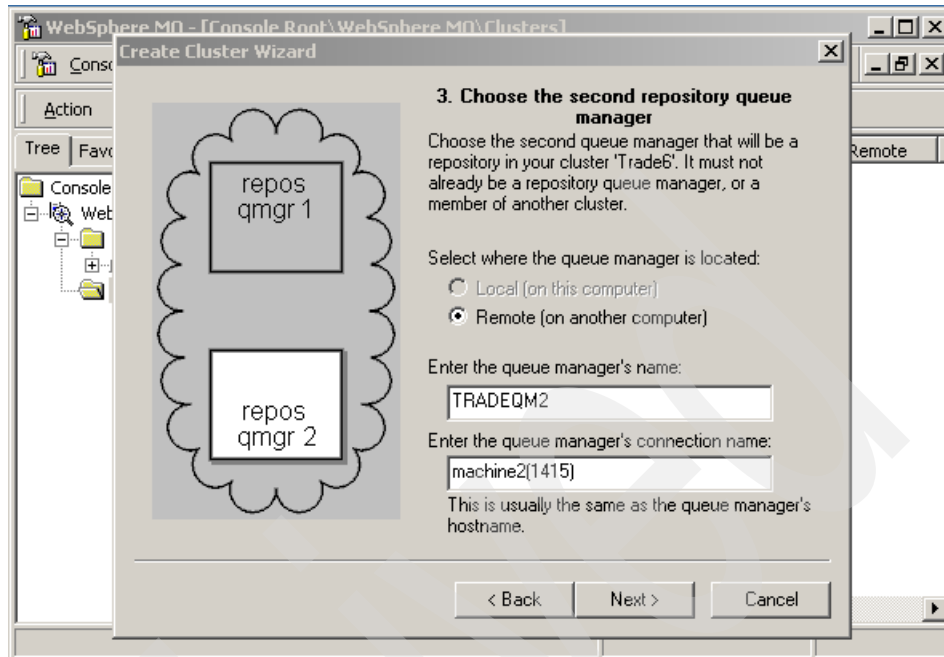


Figure 13-12 Setup second repository queue manager

7. Accept all other defaults in the wizard for channel names, verifying that the host names and ports are valid and click **Finish** to create the cluster.

Create TRADE6_RESPONSE queue on TRADEQM1

1. Now that the cluster has been created, the queues can be defined. Open the MQ Explorer and expand the **TRADEQM1** folder.
2. Right-click **Queues** and select **New -> Local Queue**.
3. On the General tab of the window that appears enter:
 - Queue name of **TRADE6_REPSONSE**
 - Default persistence of **Persistent**
4. Change to the **Cluster** tab. In here select that the queue should be **Shared in cluster** and enter **TRADE6** as the Cluster name. For Default Bind select **Not Fixed**. See Figure 13-13 on page 742.

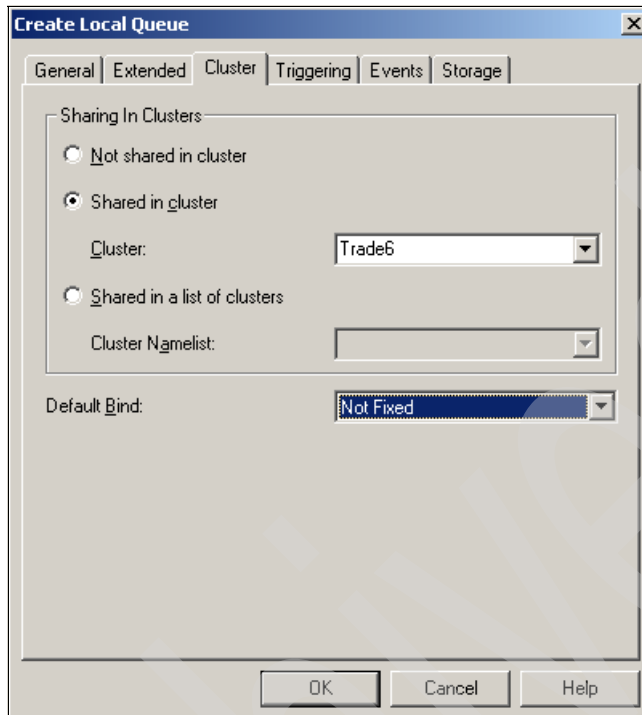


Figure 13-13 Cluster panel of creating a new queue

5. Click **OK** to create the queue.

Under **Clusters -> TRADE6 -> Queue Managers in Cluster -> TRADEQM2 -> Queues** you should now see that a new clustered queue is visible on TRADEQM1.

Create TRADE6_RESPONSE queue on TRADEQM2

Repeat the steps you just did in “Create TRADE6_RESPONSE queue on TRADEQM1” on page 741 but define the queue on **TRADEQM2**.

Once this step is complete the main part of the WebSphere MQ configuration is finished.

WebSphere Business Integration Event Broker Configuration

WebSphere Business Integration Event Broker needs DB2 and WebSphere MQ to be installed and running for it to operate. It uses configuration and runtime information stored in DB2 databases, together with queues running in WebSphere MQ, to function as a message broker. The product itself is made of three main components:

- Configuration manager
- Message broker
- Message Brokers Toolkit

These steps outline what needs to be done to get a simple publish/subscribe broker up and running. This is a very quick run through the steps needed, offering minimal explanation. For more information refer to the documentation for WebSphere Business Integration Event Broker.

1. Start DB2 on machine 3.
2. When WebSphere Business Integration Event Broker is installed, it may default to having no licenses available so a broker cannot be started. This count needs to be altered to the number of CPUs that machine 3 has and which are licensed. Open a command prompt and run the command:


```
<Event_Broker_Install_root>\bin\mqsisetcapacity -c X
```

 Where, X is the number of CPUs.
3. Launch the Message Brokers Toolkit by going to **Start -> Programs -> IBM WebSphere Business Integration Event Brokers -> Message Brokers Toolkit**.
4. The toolkit is an eclipse based set of tools which may look familiar. When it opens you are presented with the Welcome screen. (If this does not appear go to the menu option Help -> Welcome).
5. On the Welcome screen there is the option to **Create a default configuration**. Select this option as shown in Figure 13-14.

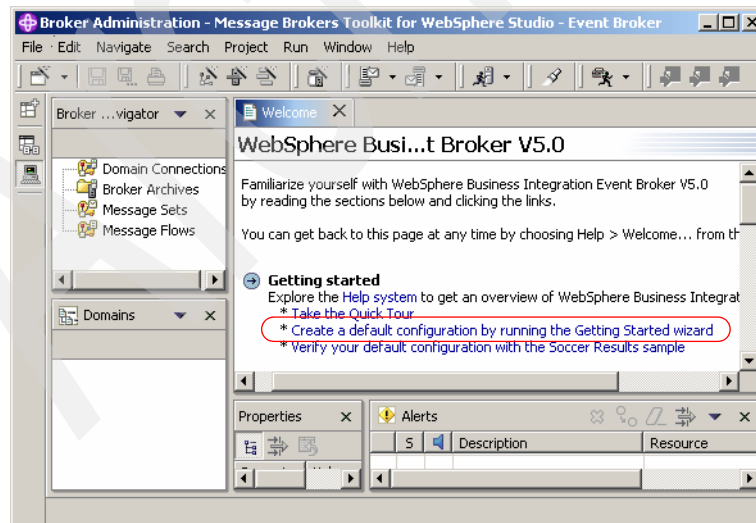


Figure 13-14 The Message Brokers Toolkit

6. On the first page make sure to use the **user name** and **password** that DB2 is installed and running under as this make installation easier. Click **Next** when done.
7. On the next page, accept the defaults for the queue manager configuration. Make sure to specify a **port number** that is not used. In this example, as with the other queue managers, the port number chosen was 1415. Click **Next**.
8. The wizard will then take you through the names of the broker and configuration managers, and their databases, accept the defaults.
9. On the final page click **Finish** to begin the process of creating all the components and starting them up. This may take a while.

Note: During this process if anything fails then experience has shown that after fixing the problem it is still best to delete the parts that succeeded before re-running the wizard such as the database tables and the project in the broker workbench. Be sure you are logged in to the operating system with the user defined in the wizard. In our configuration this was db2admin.

10. When everything has been created and started, a success message is displayed and you are returned to the Message Broker Toolkit.
11. In the Broker Administration perspective, in the bottom left view there should now be an active connection shown to the message broker. There will also be an alert message that the default Execution Group is not running. This is shown in Figure 13-15.

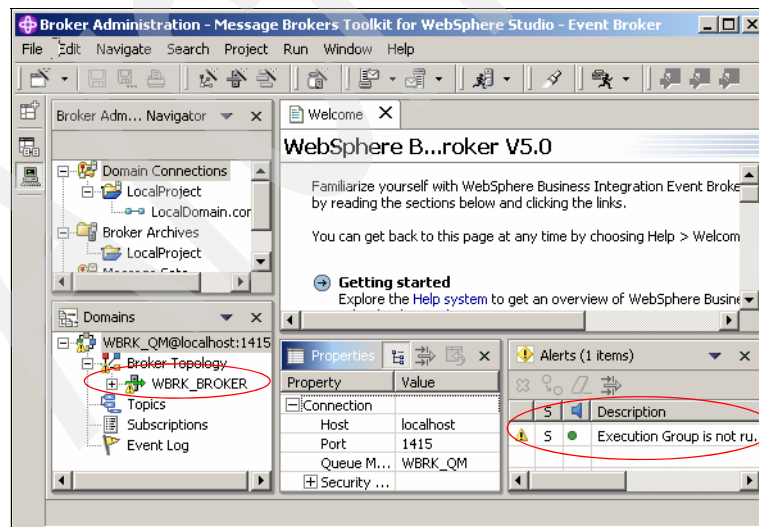


Figure 13-15 Connection to the broker

An execution group is the term used by Event Broker to group a series of message flows. A message flow is a process that describes the steps a message goes through within the broker, including the point at which it arrives. The default execution group is not running because it does not have any message flows deployed to it. A message flow is needed so that Trade 6 can use the broker. There is a sample message flow that will do for the purposes of this example. It will also reduce the number of steps needed to get the broker up and running.

12. Go to the menu option **Help -> Cheat sheets -> Preparing samples for first use**. A new window will open with the Message Brokers Toolkit.
13. We have just completed step 1, so go straight to step 2 which imports the sample into the workbench. Click the **green arrow** to run the Introduction, the **blue arrow** to skip step 1 and the **green arrow** to run step 2.
14. You are presented with a new window, select the **Soccer Results Sample** and click **Finish**.
15. Run step 3 in the Cheat sheet, creating the runtime resources. Again a window will open, select **Soccer Results Sample** and click **Next**.
16. The next panel shows the names of the queues that will be created. Select **WBRK_QM** as the queue manager and click **Finish**. You should receive a message that the create sample queues completed successfully.
17. Run the final step to deploy the soccer results message flow. After selecting the **Soccer Results Message Flow** and pressing **OK** you will be presented with options detailing which broker to publish the message flow to. Accept the defaults and click **Next**, then **Next** again.
18. Finally you will get to a page asking for the execution group. Enter the **Execution Group of Trade** and click **Next**.
19. On the last page of this client window check the box next to **Soccer Messageflows** and click **Finish**. If an error appears saying that the server cannot be started, just click **OK**.
20. Close the Cheat sheet window.
21. All that is left to do now is to start the message flow. In the Broker Administration window, under the Broker Administration Navigator, expand **Message Flows -> Soccer Messageflows -> (default)**. Right-click **SoccerPublish.msgflow** and select **Run on Server** as is shown in Figure 13-16 on page 746.

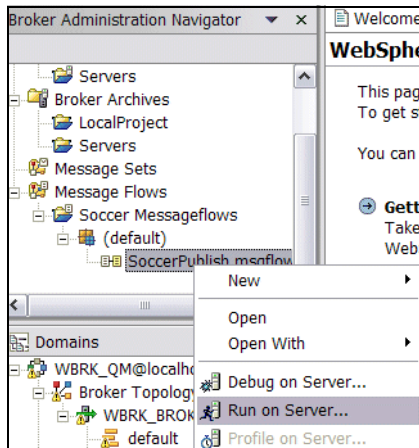


Figure 13-16 Running the soccer sample

22. After pressing **Finish** in the window that appears, you will see that the Trade execution group in the Domains view will start running. The Message Broker is now configured and ready for use. If you still have an alert for the default execution group, ignore it. The Trade execution group will be used for this example.

Important: At a later stage, for instance after a reboot, you will need to use the commands `mqsisstart configmgr` and `mqsisstart WBRK_BROKER` from the command line to get the Broker running. This should be done after DB2 has been started.

Alternatively you can start the respective services.

Trade6Redirector Installation

To complete the setup of machine 3 install the Trade6Redirector.ear file on the stand-alone application server. Trade6Redirector.zip can be found in the redbook repository. See Appendix B, "Additional material" on page 1037 for information about how to download Trade6Redirector.zip. The zip file contains a text document with installation directions.

Complete WebSphere MQ configuration for WBRK_QM

The queue manager WBRK_QM still needs to be added to the cluster and some queues defined on it. Follow these steps to complete this:

1. Open a Windows command prompt on machine 3.
2. Go to the directory <MQ_Install_root>\bin. For example:
c:\Program Files\IBM\WebSphere MQ\bin.

3. Run the command `runmqsc WBRK_QM < MQJMS_PSQ.mqsc`
This will create all of the necessary JMS queues needed for publish/subscribe.
4. Open the WebSphere MQ explorer.
5. Find **WBRK_QM** and right-click it. Select **All Tasks -> Join Cluster....**
6. In the new window that appears click **Next**.
7. When prompted enter the name of the cluster which is **TRADE6**. Click **Next**.
8. The next page will ask for details of the location of a repository queue manager. Select **Remote** and then enter the queue manager name **TRADEQM1** and the connection name of **machine1 plus the port**, for example machine1(1415). Click **Next**.
9. Upon successful communication with the remote queue manager the prompt will ask for details about the channels to link the queue managers. Accept the defaults and eventually click **Finish**.

WBRK_QM has now been added to the cluster. To verify this, take a look at the list of its queues, it should contain references to the TRADE_RESPONSE queues on the other queue managers.
10. Now that the cluster has been created the last queue can be defined. Open the MQ Explorer and expand the **WBRK_QM** folder.
11. Right-click **Queues** and select **New -> Local Queue**.
12. On the General tab of the window that appears enter:
 - Queue name of **TRADE_MAINFRAME**
 - Default persistence of **Persistent**
13. Change to the **Cluster** tab. In here select that the queue should be **Shared in cluster** and enter **TRADE6** as the cluster name. For Default Bind select **Not fixed**.
14. Click **OK** to create the queue.

This completes the setup for WebSphere MQ, there are request and response queues setup for point-to-point. The application servers will communicate with the broker through the WebSphere MQ cluster facilities. When defining the connection factories later, the name of the queue manager that the broker resides on will be used as well as the local queue manager. This allows messages to be automatically routed to the broker's queues.

WebSphere Application Server configuration

Now that the underlying messaging system has been constructed, the components within WebSphere Application Server can be put together to provide the complete platform for Trade 6 to run on.

There are four application servers that are going to be deployed. Each of them uses the messaging system in a slightly different way to reduce single points of failure in the topology. This means effective use of the scope setting for each connection factory, queue and topic. While running through these steps keep in mind the overall picture as this is a complicated configuration. Figure 13-17 shows what JMS provider objects need to be configured to get this topology to work. It should also be noted that we use a unified naming scheme and unified connection factories wherever possible.

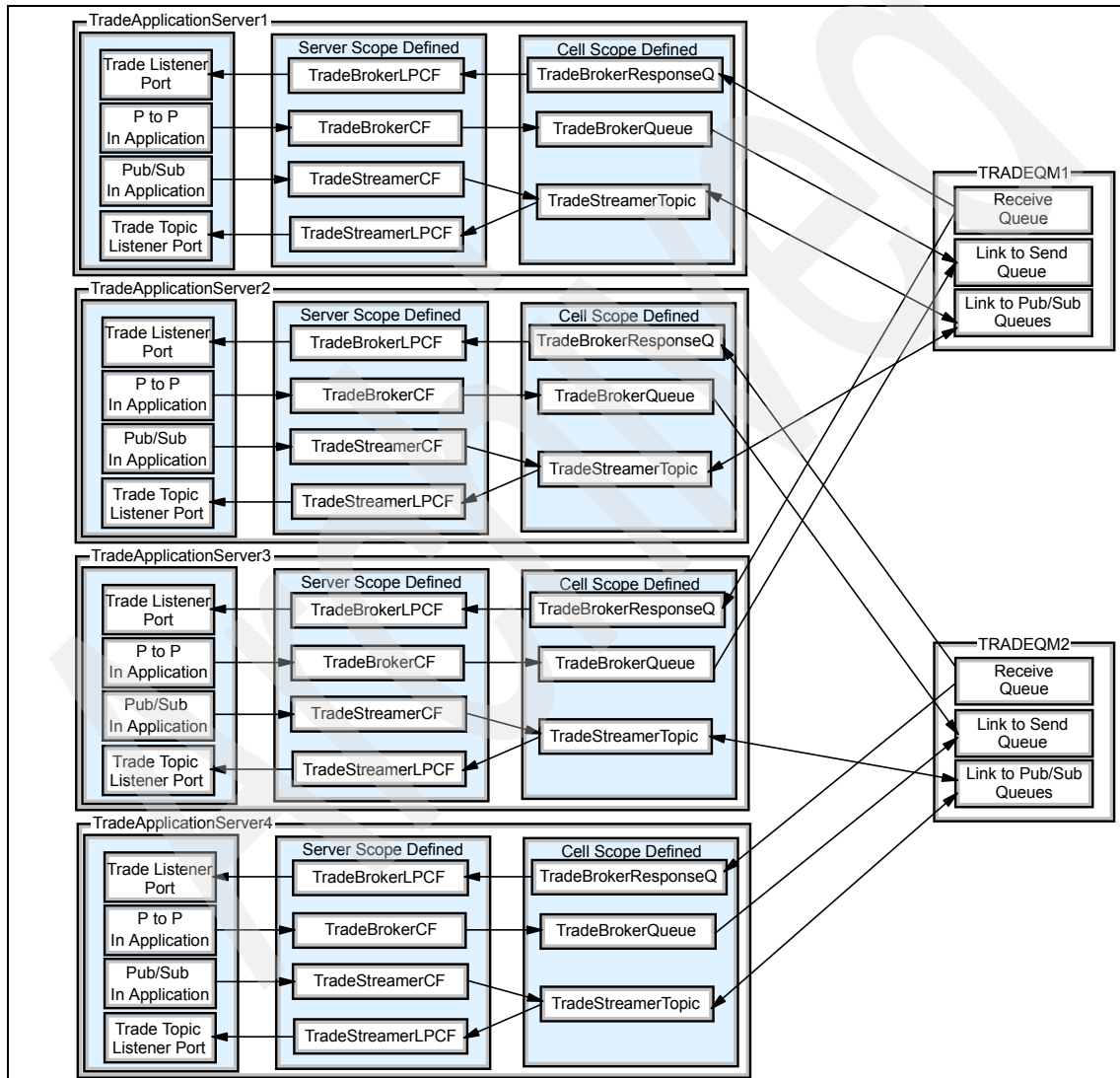


Figure 13-17 Usage of JMS provider objects in sample topology

Setup Trade 6: Part 1 - Cluster creation and JDBC resources

The creation of the cluster will be done in two stages - creating the cluster with the first server and then adding new cluster members later on. The advantage of this is that all resources defined at server scope will be copied over to the new servers.

Trade 6 needs access to a database to work. There is a script that comes with Trade 6 called `trade.jacl` which helps to setup the Trade 6 database. The script is designed to setup JMS resources for the default messaging provider as well, so there will be resources installed which are not relevant for this scenario and can be removed using the Administrative Console after executing the script.

Use the following command to run the script:

```
<WebSphere_Path>\bin\wsadmin -f trade.jacl configure
```

Be sure to run `setupCmdLine.bat` first from the `bin` directory of your WebSphere installation. See 8.8, “Installing and configuring Trade 6” on page 436 for detailed instructions.

The following example illustrates the settings that were selected during the script's execution under our environment which uses DB2 as the database. The defaults are used when no choice is specified below the setting.

Important: A database user name longer than 12 characters is not recommended. If the user name is more than 12 characters, it cannot be used as an authentication entry for WebSphere MQ connection factories.

Example 13-1 Trade 6 script settings

Global security is (or will be) enabled (true|false) [false]:

Is this a cluster installation (yes|no) [no]:

yes

Have all nodes been federated and network connectivity verified? (yes|no) [yes]:

Please enter the cluster name [TradeCluster]:

Select the desired node [app1Node]:

Please enter the cluster member name [TradeServer1]:

Add more cluster members (yes|no) [yes]:

no

Select the backend database type (db2|oracle) [db2]:

Please enter the database driver path

[c:/sql1lib/java/db2jcc.jar;c:/sql1lib/java/db2jcc_license_cu.jar;c:/sql1lib/java/db2jcc_license_cisuz.jar]:
\${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar;\${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar

Please enter the database name [tradedb]:

Please enter the DB2 database hostname [localhost]:
db

Please enter the DB2 database port number [50000]:

Please enter the database username [db2admin]:

Please enter the database password [password]:
password

Note: Be sure to check the entire output. Even if the last task is completed successfully, you may need to scroll up to check that there were no previous errors.

You may want to test the database connection in the Administrative Console under **Resources -> JDBC Providers -> DB2 Universal JDBC Driver Provider (XA) -> Data sources**. Be sure to select the **Cell scope level** after selecting JDBC Providers. If you receive an error in connecting to the database you may probably need to set the DB2UNIVERSAL_JDBC_DRIVER_PATH variable under **Environment -> WebSphere Variables** at the **Node** scope level.

Because Trade 6 uses the DB2 universal JDBC Type 4 driver, it is usually not necessary to configure the DB2 clients on the application server systems any more (as was needed for previous versions of Trade). If however, for some reason, you have problems accessing the database, then you should follow the steps outlined in “DB2 clients” on page 439 which should solve your connection problems.

The script sets up a Java 2 connector authentication user name and password for the database for which db2admin was used in Example 13-1 on page 749. It is of particular interest because this authentication entry is used in this example for database access and access to WebSphere MQ queue managers. The authentication entry can be inspected using the Administrative Console under **Security -> Global security -> JAAS Configuration -> J2C Authentication data -> TradeDataSourceAuthData**.

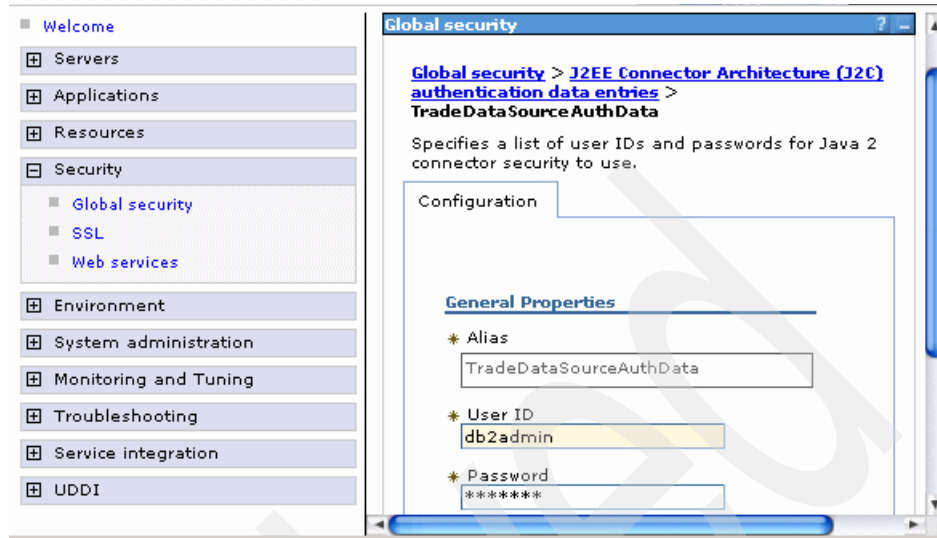


Figure 13-18 TradeDataSourceAuthData entry

Remove unnecessary resources

The trade.jacl script was written for the new default messaging provider in WebSphere Application Server V6. Thus, it creates resources that are not used in this example. These resources should be removed to avoid possible JNDI name conflicts and to keep a clean environment. It is recommended that the following steps be performed to remove the unused resources from the environment:

1. Remove the following resources from the default messaging provider at the cell scope level under **Resources -> JMS Providers -> Default messaging**.
 - TradeBrokerQCF Queue Connection Factory
 - TradeStreamerTCF Topic Connection Factory
 - TradeBrokerQueue JMS queue
 - TradeStreamerTopic JMS topic
 - TradeBrokerMDB JMS activation specification
 - TradeStreamerMDB JMS activation specification
2. Remove the TradeCluster bus from **Service integration -> Buses**.
3. Remove MEDataSource from **Resources -> JDBC providers -> DB2 Universal JDBC Driver Provider -> Data sources**.
4. Remove Policy for ME0 from **Servers -> Core groups -> Core group settings -> DefaultCoreGroup > Policies**.

Configure JMS resources: Part 1

The first task is to identify how many JMS components need to be defined within the cell.

- ▶ **Point-to-point connection factory**

By looking at Figure 13-8 on page 733 it is possible to see that for point-to-point messaging each of the application servers need to access the WebSphere MQ cluster in a different way. This is because each possible combination of two queue managers and two forms of communication needs to be covered (CLIENT and BINDINGS). This is achieved by creating all the relevant connection factory objects at the Server scope, making them visible only to that server.

- ▶ **Publish/subscribe connection factory**

For publish/subscribe messaging, the MDB has a durable subscription. Each application server needs to receive a copy of any published message so connection factories cannot be shared. Four application servers means there needs to be four durable subscriptions, each made unique by the client ID set on the connection factory (as the MDB name is the same - cloned application). This means defining the connection factory objects at the Server scope level as well. More information about this can be found in “Client ID and Enable Clone support for publish/subscribe” on page 715.

- ▶ **Destinations**

Finally the queue destination and topic destination contain no information that is specific to a particular application server, so these can be configured at Cell level.

Follow these steps below to create the necessary listener ports and JMS provider objects on TradeServer1.

After completing these steps, TradeServer1 will be used as a template to create the other servers and the final configuration changes will be made. This includes configuring dedicated connection factories for the listener ports to use. This allows for specific configuration of a connection factory for listener port usage and makes defining the size of the connection and session pools simpler.

1. Open the WebSphere Administrative Console, expand **Resources -> JMS Providers** and select **WebSphere MQ**.
2. Set the scope to Server **TradeServer1** (on Node app1) and click **Apply**.

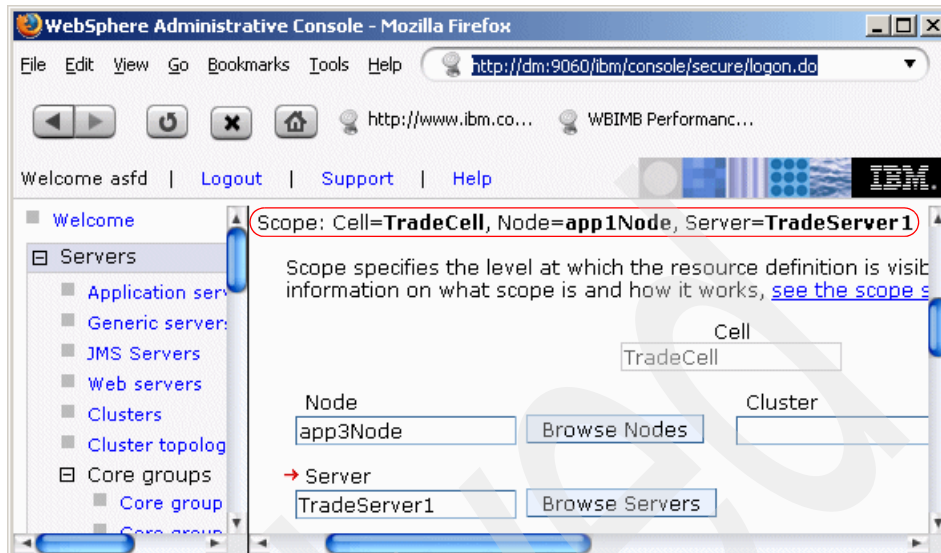


Figure 13-19 Set the scope level for app1, TradeServer1

3. Select **WebSphere MQ connection factories**.
4. Create two new connection factories using the information from Table 13-3 and Table 13-4. One is for use by the listener port. If a field is not specified then use its default value.

Table 13-3 TradeBrokerCF

Field	Value
Name	TradeBrokerCF
JNDI name	jms/TradeBrokerCF
Queue manager	TRADEQM1
Transport type	BINDINGS
Component-managed authentication alias	TradeDataSourceAuthData
Container-managed authentication alias	TradeDataSourceAuthData
XA enabled	Checked (True)

Table 13-4 TradeBrokerListenerPortCF

Field	Value
Name	TradeBrokerListenerPortCF

Field	Value
JNDI name	jms/TradeBrokerLPCF
Queue manager	TRADEQM1
Transport type	BINDINGS
Component-managed authentication alias	TradeDataSourceAuthData
Container-managed authentication alias	TradeDataSourceAuthData
Enable XA	Checked (True)

5. Go back into TradeBrokerListenerPortCF by clicking **TradeBrokerListenerPortCF**. Select **Connection pool** from the Additional Properties pane.
6. Change Minimum connections to **1** and Maximum connections to **2**. The listener port will need only one connection. Click **OK**.
7. Select **Session pools**, again from the Additional Properties.
8. Change Minimum connections to **1** and Maximum connections to **5**. The listener port will only need at most 5 sessions as this will be defined by the maximum sessions setting on the listener port. Click **OK** and then **OK** again.
9. Go back to the **WebSphere MQ connection factories**.
10. Create two new connections using the information in Table 13-5 on page 755 and Table 13-6 on page 755. These factories will be used for publish/subscribe messaging and for the respective listener port. If a field is not specified then use its default value.

Table 13-5 TradeStreamerListenerPortCF

Field	Value
Name	TradeStreamerListenerPortCF
JNDI name	jms/TradeStreamerLPCF
Transport Type	BINDINGS
Component-managed authentication alias	TradeDataSourceAuthData
Container-managed authentication alias	TradeDataSourceAuthData
Queue manager	TRADEQM1
Broker queue manager	WBRK_QM
Broker publication queue	SOCCER_PUBLICATION This is the name of the queue that was setup on WBRK_QM. It is being monitored by the broker based on the sample message flow.
Broker subscription queue	SOCCER_SUBSCRIPTION
Broker version	Advanced
Enable clone support	Uncheck (False) Although there is use of durable subscriptions, this connection factory is only being used by one application server in the cluster and so there is no issue with MDB listener ports starting up with the same client IDs.
Client ID	TradeServer1 This unique ID is what allows multiple durable subscriptions on the same topic from within the server cluster. This needs to be different for each server scope.
Enable XA	Uncheck (False) There is no requirement for 2PC on this object.

Table 13-6 TradeStreamerCF

Field	Value
Name	TradeStreamerCF
JNDI name	jms/TradeStreamerCF

Field	Value
Transport type	BINDINGS
Component-managed authentication alias	TradeDataSourceAuthData
Container-managed authentication alias	TradeDataSourceAuthData
Queue manager	TRADEQM1
Broker queue manager	WBRK_QM
Broker publication queue	SOCCER_PUBLICATION
Broker subscription queue	SOCCER_SUBSCRIPTION
Broker version	Advanced
Enable clone support	Uncheck (False)
Client ID	TradeServer1
Enable XA	Uncheck (False).

11. Setup the connection and session pools for this resource by repeating steps 5 on page 754 to 8 on page 754 but for **TradeStreamerListenerPortCF** this time.
12. Next go back to **WebSphere MQ messaging provider**. Reset the scope to the **Cell level**. This is done by removing app1 and TradeServer1, then pressing **Apply**.
13. Click **WebSphere MQ queue destinations**. Two queue destinations need to be created, one that points to the send queue that will go to the back-end process, Trade6Redirector, and one that points to the response queue that will be used by the MDB.
14. Create two new queue destinations using the information from Table 13-7 on page 757 and Table 13-8 on page 757. Where no value is given, use the defaults.

Table 13-7 TradeBrokerQueue

Field	Value
Name	TradeBrokerQueue
JNDI name	jms/TradeBrokerQueue
Persistence	PERSISTENT
Base queue name	TRADE_MAINFRAME

Table 13-8 TradeBrokerResponseQueue

Field	Value
Name	TradeBrokerResponseQueue
JNDI name	jms/TradeBrokerResponseQueue
Persistence	PERSISTENT
Base queue name	TRADE6_RESPONSE

15. Go back to the **WebSphere MQ messaging provider** and click **WebSphere MQ topic destinations**.

16. Create a new topic destination using the information from Table 13-9.

Table 13-9 TradeStreamerTopic

Field	Value
Name	TradeStreamerTopic
JNDI name	jms/TradeStreamerTopic
Persistence	PERSISTENT
Base topic name	TradeStreamerTopic

17. Now we need to create the listener ports. Go to **Servers -> Application servers -> TradeServer1 -> Messaging -> Message Listener Service -> Listener Ports**.

18. Two listener ports are needed, one for point-to-point and one for publish/subscribe. Create two new listener ports using the information from Table 13-10 on page 758 and Table 13-11 on page 758. Where no value is given for a field use the default.

Table 13-10 tradeport

Field	Value
Name	tradeport
Connection factory JNDI name	jms/TradeBrokerLPCF
Destination JNDI name	jms/TradeBrokerResponseQueue
Maximum sessions	5
Maximum retries	10

Table 13-11 tradetopicport

Field	Value
Name	tradetopicport
Connection factory JNDI Name	jms/TradeStreamerLPCF
Destination JNDI name	jms/TradeStreamerTopic
Maximum sessions	1
Maximum retries	10

19. **Save** the configuration.
20. The install path for IBM WebSphere MQ needs to be set if it is not set already. Go to **Environment -> WebSphere Variables**. Change the scope to the **app1 Node** and then set the **MQ_INSTALL_ROOT** variable. Do the same for **app2**.
21. TradeServer1 is now set up and ready to run.
22. At this point it is recommended that you test the configuration on one server following the directions in “Configuring and installing the application” on page 762. Only TradeServer1 is configured at this stage but it is enough to verify that the messaging system is working and resolve any errors before the entire topology is setup.

Trade 6 configuration: Part 2 - Setup additional cluster members

It is now time to create the other three servers. TradeServer1 will now be used as a template, reducing the effort to setup all the JMS resources.

1. In the WebSphere Administrative Console go to **Servers -> Clusters -> TradeCluster -> Cluster members -> New**.
2. Enter the name **TradeServer2**, select **app1** as the node. Make sure that **Generate Unique HTTP Ports** is checked, then click **Apply**.

3. Repeat this for servers **TradeServer3** and **TradeServer4** which need to be created on **app2**.
4. Once all the servers are in the list click **Next** and then **Finish**.
5. Each of the new application servers needs its Web container transport ports added to the default_host virtual host before they can accept requests. To find the port numbers, go to **Servers -> Application servers -> <AppServer_Name> -> Web Container Settings -> Web container transport chains**. Verify the port number for **WCInboundDefault**.

Go to each application server in turn and write down the ports.

Then go to **Environment -> Virtual Hosts -> default_host -> Host Aliases**. Add the ports if they are not listed.

Configure JMS resources: Part 2 - Adjust JMS settings for new cluster members

The final step in this setup is to change the connection factories for TradeServer2, TradeServer3, and TradeServer4 to point to the correct queue manager using the correct transport type. As TradeServer1 was used as a template for creating these new application servers, all of its resources have come across as well. The listener ports should be left unchanged. Only the connection factory settings need to be changed.

1. In the scope of **app1/TradeServer2** on the **WebSphere MQ messaging provider** change the fields outlined in Table 13-12 to Table 13-15 on page 760 for each connection factory:

Table 13-12 TradeBrokerCF

Field	Value
Name	TradeBrokerCF
Queue Manager	TRADEQM2
Transport Type	CLIENT
Host	<machine2>
Port	1415

Table 13-13 TradeBrokerListenerPortCF

Field	Value
Name	TradeBrokerListenerPortCF
Queue Manager	TRADEQM2

Field	Value
Transport Type	CLIENT
Host	<machine2>
Port	1415

Table 13-14 TradeStreamerCF

Field	Value
Name	TradeStreamerCF
Client ID	TradeServer2

Table 13-15 TradeStreamerListenPortCF

Field	Value
Name	TradeStreamerListenerPortCF
Client ID	TradeServer2

- In the scope of **app2/TradeServer3** for the **WebSphere MQ Provider** change the fields shown in Table 13-16 to Table 13-19 on page 761 for each connection factory:

Table 13-16 TradeBrokerCF

Field	Value
Name	TradeBrokerCF
Queue Manager	TRADEQM1
Transport Type	CLIENT
Host	<machine1>
Port	1415

Table 13-17 TradeBrokerListenerPortCF

Field	Value
Name	TradeBrokerListenerPortCF
Queue Manager	TRADEQM1
Transport Type	CLIENT
Host	<machine1>

Field	Value
Port	1415

Table 13-18 *TradeStreamerCF*

Field	Value
Name	TradeStreamerCF
Queue Manager	TRADEQM2
Client ID	TradeServer3

Table 13-19 *TradeStreamerListenerPortCF*

Field	Value
Name	TradeStreamerListenerPortCF
Queue Manager	TRADEQM2
Client ID	TradeServer3

3. In the scope of **app2/TradeServer4** for the **WebSphere MQ messaging provider** change the fields outlined in Table 13-20 to Table 13-23 on page 762 for each connection factory:

Table 13-20 *TradeBrokerCF*

Field	Value
Name	TradeBrokerCF
Queue Manager	TRADEQM2
Transport Type	BINDINGS

Table 13-21 *TradeBrokerListenerPortCF*

Field	Value
Name	TradeBrokerListenerPortCF
Queue Manager	TRADEQM2
Transport Type	BINDINGS

Table 13-22 *TradeStreamerCF*

Field	Value
Name	TradeStreamerCF

Field	Value
Queue Manager	TRADEQM2
Client ID	TradeServer4

Table 13-23 TradeStreamerListenerPortCF

Field	Value
Name	TradeStreamerListenerPortCF
Queue Manager	TRADEQM2
Client ID	TradeServer4

4. If not already defined, you may need to set the **DB2UNIVERSAL_JDBC_DRIVER_PATH** variable under **Environment -> WebSphere Variables** at the node scope for **app1** and **app2** respectively.

After saving the configuration, the JMS resources are setup for the application.

Configuring and installing the application

Trade 6 is setup to use non-durable message-driven beans by default. In order to change this, the deployment descriptors for the message-driven beans need to be changed slightly.

Using IBM Rational Application Developer V6 or the Application Server Toolkit V6, import the trade.ear file and switch to the J2EE perspective. Expand **EJB Projects -> tradeEJB**. Double-click **Deployment Descriptor: TradeEJBs**. Select the **Bean** tab. Select **TradeStreamerMDB** from the Bean list. In the Activity Configuration panel on the right hand side click **Add....** The window shown in Figure 13-20 on page 763 should appear. Enter a Name of **subscriptionDurability** and a Value of **Durable**. Click **Finish**, save your changes and export the modified Trade6 ear.

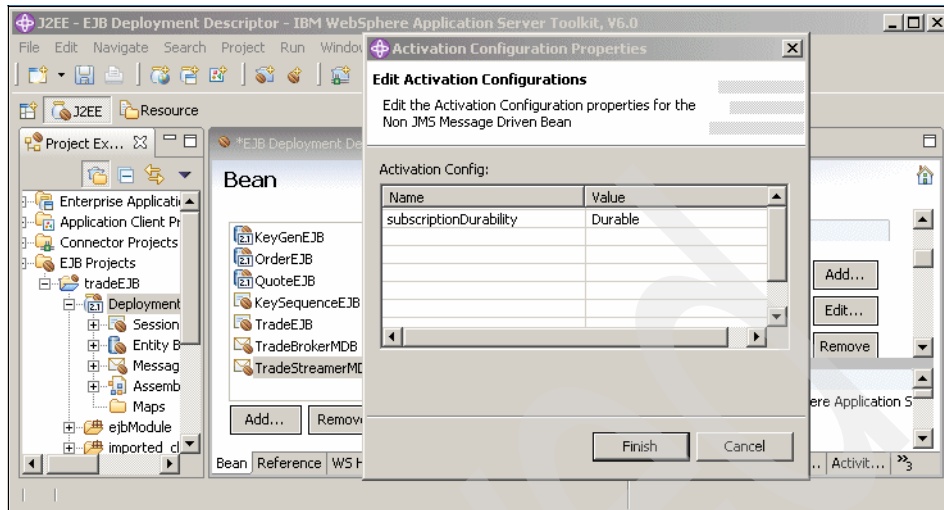


Figure 13-20 Modifying trade.ear for a durable subscription

Follow these steps to use the WebSphere Administrative Console to install the modified Trade 6 EAR file:

1. Select **Applications -> Install New Application**. Specify the modified Trade 6 EAR created above and click **Next** or **Continue** until the Step 2: Map modules to servers panel is displayed.
2. On the Step 2 panel, verify that the application is mapped to the correct cluster and map it also to your Web server(s), if you want to access the application through the HTTP server(s). Click **Next**.
3. On the Step 3: Provide options to perform the EJB Deploy panel select the database type according to your environment and click **Next**.
4. Change the Bindings for TradeBrokerMDB to **Listener Port** and enter the Name **tradeport**.

Do the same for TradeStreamMDB, using the Name **tradetopicport** and click **Next**.

5. Accept the defaults until Step 10: Map resource references to resources. Click **Continue** if you receive Application Resource Warnings.

Change the TradeBroker and TradeStream connection factory JNDI names to reflect the unified connection factory naming we have used thus far:

- Change jms/TradeBrokerQCF to **jms/TradeBrokerCF** for both the TradeEJBs and TradeWeb modules.
- Change jms/TradeStreamTCF to **jms/TradeStreamCF** for both the TradeEJBs and TradeWeb modules.

6. Accept the defaults for the remainder of the installation.

Testing the application

The queue managers and message brokers should already be started. If you receive any errors when starting up the application servers verify that the WebSphere MQ cluster is working by looking at the channel and listener status in the WebSphere MQ Explorer. Also, do not forget to start the Trade6Redirector application to do the back-end processing.

The Trade 6 application defaults to using Synchronous as the order process mechanism on each server. For this reason change the Trade 6 configuration on each of the servers. For each server go to

`http://<server_host>:908X/trade/config`

and change the following configuration parameters:

- ▶ Order processing mode = **Asynchronous_2-Phase**
- ▶ Enable Operational Trace = **True**
- ▶ Enable Full Trace = **True**

Click **Update config** to complete.

Trade 6 is now up and running. There are many aspects of the fail over to try out. Here are a couple of tests to start off with:

- ▶ Run the Web primitives `PingServletToMDBQueue` and `PingServletToMDBTopic`. Check the `SystemOut.log` of the servers to verify the message has been delivered.
- ▶ Stop the listener ports on the servers in the **Servers -> Application servers -> <TradeServerX> -> Messaging -> Message Listener Service -> Listener Ports** menu. Log in to Trade 6 and place some orders. The orders will not be completed as the MDBs are stopped. Use the WebSphere MQ Explorer to observe workload management delivering the messages to the queues. Start the listener ports back up again and on the next visit you should receive a notification that the orders were completed.
- ▶ Stop the listener ports on one of the servers. Place some orders in Trade 6. All the started servers will receive the updated stock price in their logs except the stopped server. Start its message listener service back up and all the publications it missed will be delivered.

13.6 Monitoring performance with Tivoli Performance Viewer

This section assumes an understanding of how Tivoli Performance Viewer works. If you need to learn more first then go to Chapter 14, “Server-side performance and analysis tools” on page 769.

The Performance Monitoring service within each application server provides statistics relevant to WebSphere MQ in the following performance modules:

- ▶ MDB executions including average response time of `onMessage` method
- ▶ Usage of JCA Connection Pools
- ▶ Message listener threads

The available counters when using Trade 6 are shown in Figure 13-21 on page 766. Make sure the appropriate counters are enabled in the PMI Custom settings (**Monitoring and Tuning -> Performance Monitoring Infrastructure (PMI) -> <AppServer_Name> -> Custom** or Tivoli Performance Viewer cannot display them.

To display them, go to **Monitoring and Tuning -> Performance Viewer -> Current Activity -> <AppServer_Name> -> Performance Modules**.

Normally one or two values are inspected at a time for a running application. Figure 13-21 on page 766 has over a dozen values selected, but it is more practical to review a few related values at a time. For example, you may want to inspect the size of JCA Connection Pools associated with all the JMS connection factories an application uses.

For more information about the Tivoli Performance Viewer, please refer to 14.3, “Using Tivoli Performance Viewer” on page 790.

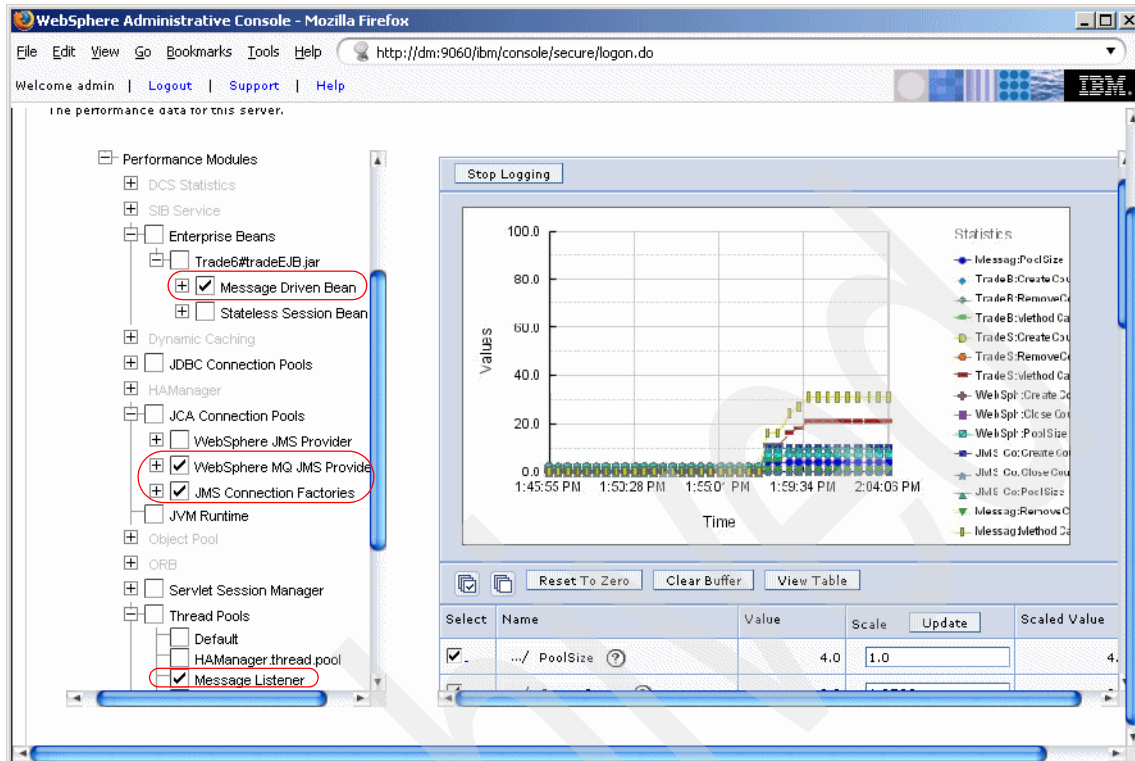


Figure 13-21 Counters for monitoring WebSphere MQ

13.6.1 What do the counters under JCA Connection Pools mean?

A JCA resource adapter is associated with WebSphere MQ. The underlying implementations of `javax.jms.Session` and `javax.jms.Connection` are associated with a resource adapter in WebSphere Application Server V6.

The implementation details are beyond the scope of this redbook. However, what is important to know as an architect or administrator is that if these pools are overstrained, WebSphere MQ connection factory session pools, connection pools, the message listener service, or the application itself may need to be tuned to alleviate a high messaging load. Thus, monitoring the JCA Connection Pools is in effect monitoring connections to the WebSphere MQ messaging provider.

For more information about the JCA resource adapters see 10.3, “Messaging in the J2EE Connector Architecture” in *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.



Part 6

Performance monitoring, tuning, and coding practices

Server-side performance and analysis tools

In a production environment, performance and availability of Web applications are critical. In this chapter, we describe how a production environment should be monitored, the types of data available for monitoring, the tools available within WebSphere Application Server to display that data, and tools included to provide guidance on how to optimize performance.

This chapter discusses the following topics:

- ▶ The dimensions of monitoring
- ▶ Performance Monitoring Infrastructure
- ▶ Using Tivoli Performance Viewer
- ▶ Other performance monitoring and management solutions
- ▶ Developing your own monitoring application
- ▶ Request Metrics
- ▶ Performance Advisors
- ▶ Dynamic Cache Monitor
- ▶ Monitoring the IBM HTTP Server
- ▶ Log Analyzer
- ▶ Application management

14.1 The dimensions of monitoring

Performance problems can be encountered almost anywhere. The problem can be network and hardware related, back-end system related; it can be actual product bugs, or quite often, application design issues.

Understanding the flow used to diagnose a problem helps to establish the monitoring that should be in place for your site to detect and correct performance problems. The first dimension is the "end-user view," the black box view of your Web site. This is an external perspective of how the overall Web site is performing from an end user's point of view and identifies how long the response time is for an end user. From this black box perspective, it is important to understand the load and response time on your site. To monitor at this level, many industry monitoring tools allow you to inject and monitor synthetic transactions, helping you identify when your Web site experiences a problem.

The second step is to understand the basic health of all the systems and network that make an end user request. This is the "external" view, which typically leverages tools and utilities provided with the systems and applications running. In this stage, it is of fundamental importance to understand the health of *every system* involved - including Web servers, application servers, databases, back-end systems, etc. If any of the systems has a problem, it may have a rippling effect and cause the "servlet is slow" problem.

This dimension corresponds to the "what resource is constrained" portion of the problem diagnosis. To monitor at this level, WebSphere provides PMI instrumentation and the Tivoli Performance Viewer as a starting point. There are also several industry tools built using PMI instrumentation that provide 24x7 monitoring capabilities.

The third dimension is the application view. This dimension actually understands the application code that is satisfying the end user request. This dimension understands that there are specific servlets that are accessing session beans, to entity CMP beans, to a specific database, etc. This dimension typically comes into play in the in-depth internal understanding of who is using the resource. Typically at this stage, some type of time trace through the application, or thread analysis under load conditions techniques are deployed to isolate areas of the application, and particular interactions with back-end systems or databases that are especially slow under load. WebSphere provides the Request Metrics technology as a starting point. In many cases, you start moving into using a lot of the development tools provided, such as IBM Rational Application Developer V6.0.

14.1.1 Overview: Collecting and displaying application server data

Table 14-1 shows the types of data that can be collected, the required actions to collect it, and how to view it.

Table 14-1 Performance data collection and viewing

Type of data	Steps/methods of configuration	Viewed with
Performance Monitoring Infrastructure service provides performance data for system resources, WebSphere Application Server, and a customer's application across all transactions. (PMI also includes JVMPI data.)	<ol style="list-style-type: none">1. Set at application server level and Node Agent:<ul style="list-style-type: none">– Administrative Console (define the monitored statistic set, such as Basic)– wsadmin2. Configure instrumentation level:<ul style="list-style-type: none">– Tivoli Performance Viewer: select Monitoring and Tuning -> Performance Viewer -> Current Activity -> <AppServer_Name>– wsadmin	<ul style="list-style-type: none">► Tivoli Performance Viewer► Monitoring tools using interfaces such as JMX or Performance Servlet
Request Metrics provides response time data for each individual transaction such as time spent in the Web server, Web container, EJB container, JMS processing, and the back-end database.	Set at cell level in: <ul style="list-style-type: none">– Administrative Console– wsadmin	<ul style="list-style-type: none">► System.out log file► http_plugin.log file► Monitoring tools using the ARM interface

14.2 Performance Monitoring Infrastructure

The second stage of monitoring as described in 14.1, “The dimensions of monitoring” on page 770 was understanding the basic health of all the systems and network that make up an end user request. For the WebSphere environment, we provide the Performance Monitoring Infrastructure APIs to capture performance data with minimal performance impact to incorporate that data into an overall monitoring solution.

The *Performance Monitoring Infrastructure* (PMI) provides a set of APIs to obtain performance data for system resources, WebSphere Application Server queues, and actual customer application code.

PMI uses a client-server architecture. The server collects performance data in memory within the WebSphere Application Server. This data consists of counters such as servlet response time and data connection pool usage. A client can then retrieve that data using a Web client, a Java client, or a Java Management Extension (JMX) client. A client is an application that retrieves performance data from one or more servers and processes the data. Clients can include:

- ▶ Graphical user interfaces (GUIs) that display performance data in real time.
- ▶ Applications that monitor performance data and trigger different events according to the current values of the data.
- ▶ Any other application that needs to receive and process performance data.

PMI complies with the Performance Data Framework of the J2EE Management Specification. It is composed of components for collecting performance data on the application server side and components for communicating between runtime components and between the clients and servers. The primary PMI components and related Management Beans (MBeans) are illustrated in Figure 14-1.

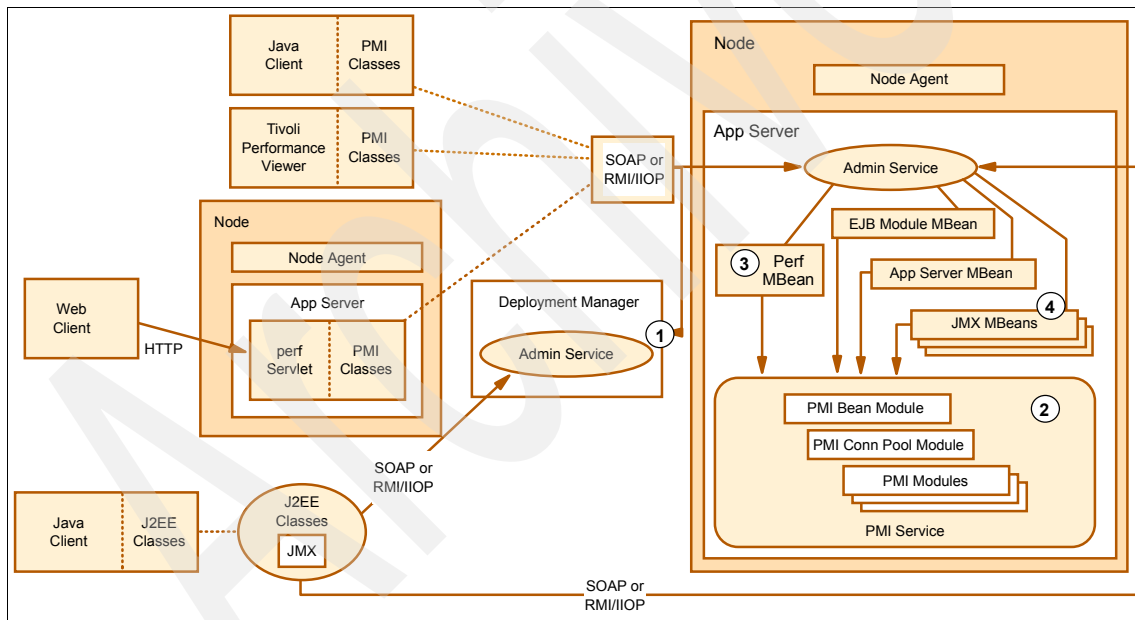


Figure 14-1 Performance monitoring components

1. The Tivoli Performance Viewer contacts the administrative service of the Deployment Manager (1) to get a list of nodes, servers and MBeans for the entire cell.

2. The PMI service in the application servers, consisting of PMI modules for collecting performance data and methods for instrumenting and retrieving the data from the runtime components.
3. PerfMBean, a JMX management bean used to extract performance data from the PMI modules.
4. Extensions of the standard JMX MBeans (used for managing components and settings) to support management of performance data. This enables the retrieval of performance data via JMX. See 3.2, “Java Management Extensions (JMX)” in the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451, for a description of the JMX framework.

The J2EE classes and PMI classes can use either the RMI over IIOP or SOAP protocol to communicate with the administrative service. In a single-server environment, the classes connect to the administrative service of each individual application server in order to collect performance data. In a Network Deployment environment, the client connects to the Deployment Manager to retrieve a list of nodes, servers and MBeans in the cell. Performance data retrieval is subsequently performed in the same way for the two environments.

Each piece of performance data has two components: a static component and a dynamic component. The dynamic component consists of a name and an ID to identify the data, as well as other descriptive attributes that assist the client in processing and displaying the data. The dynamic component consists of information that changes over time, such as the current value of a counter and the time stamp associated with that value.

14.2.1 Performance data classification

PMI provides several different metrics. Each of these metrics is classified into one of the following five types to provide some standardization within the infrastructure. This classification is based on J2EE management specifications and is important to understand when developing a monitoring tool.

- ▶ **Count statistic:** Specifies standard count measurements. It consists of a single numeric value and is used to represent data such as counts and sizes. Examples of count statistics include number of times beans were created, number of calls retrieving an object from the pool, and used memory in the JVM runtime.
- ▶ **Boundary statistic:** Specifies standard measurements of the upper and lower limits of the value of an attribute. This classification is currently not being used by PMI.
- ▶ **Range statistic:** Specifies standard measurements of the lowest and highest values an attribute has held as well as its current value. These values can be

used for obtaining the number of concurrent invocations to a call method, average number of threads in a pool, or the number of requests that are concurrently processed by the ORB.

- ▶ **Bounded range statistic:** Extends the Range statistic and Boundary statistic interfaces and provides standard measurements of a range that has fixed limits. Examples of use of the Bounded range statistic interface are the number of free connections in the J2C pool, total memory in JVM runtime, and average number of threads in pool.
- ▶ **Time statistic:** Specifies standard timing measurements for a given operation. Examples of time statistics are average response time in milliseconds on the bean methods (home, remote, local) and average connection time in milliseconds until a connection is granted.

14.2.2 Performance data hierarchy

Performance data is provided in a centralized hierarchy of the following objects to help provide some logical ordering:

- ▶ **Node:** A node represents a physical machine in the WebSphere cell. This is where the Node Agent resides in a Deployment Manager environment.
- ▶ **Server:** A server is a functional unit that provides services to the clients over a network. No performance data is collected for the server itself.
- ▶ **Module:** A module represents a resource category for which performance data is collected. As an example, these are Enterprise JavaBeans, database connection pools, J2C connectors, JVM runtime, Object Request Broker, relational resource adapter, servlet session manager, thread pools, Transaction Manager, and Web applications.
- ▶ **Submodule:** A submodule represents a fine granularity resource category under the module. Submodules themselves can contain submodules. An example of a submodule is the ORB thread pool, which is a fine granularity resource category under the thread pool module.
- ▶ **Counter:** A counter is a data type used to hold performance information for analysis. Examples of counters include the number of active enterprise beans and the time spent responding to a servlet request.

Each resource category (module) has a related set of counters. The counters contain values for performance data that can have an impact on system performance.

Modules can have instances, which are single instantiations of an object of a class. Counters can be assigned to any number of modules and instances. Figure 14-2 on page 775 shows how this looks like in Tivoli Performance Viewer.

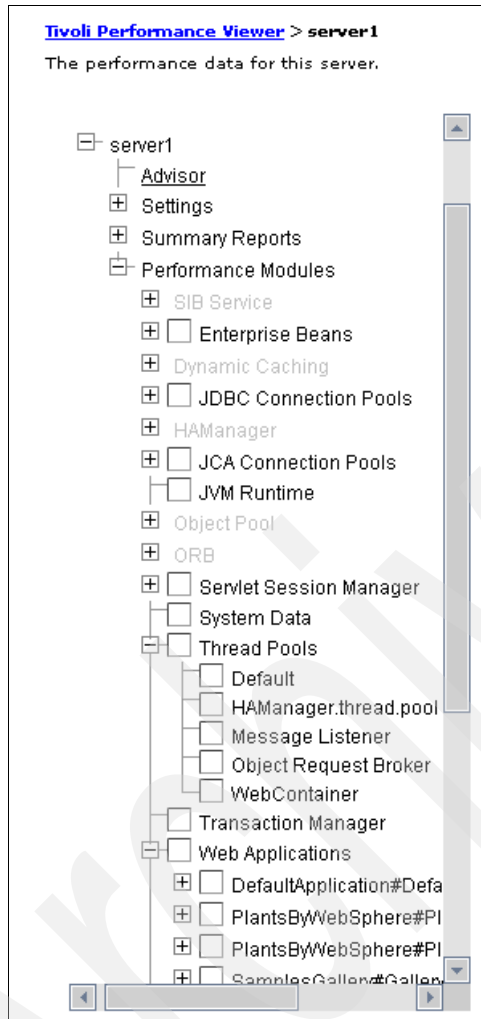


Figure 14-2 Tivoli Performance Viewer - performance data selection

Figure 14-3 on page 776 shows the counter Avg Method RT enabled to both the enterprise beans module and the methods of the Container1.Bean1 instance. It also shows a hierarchy of data collections that are organized for reporting to the Tivoli Performance Viewer.

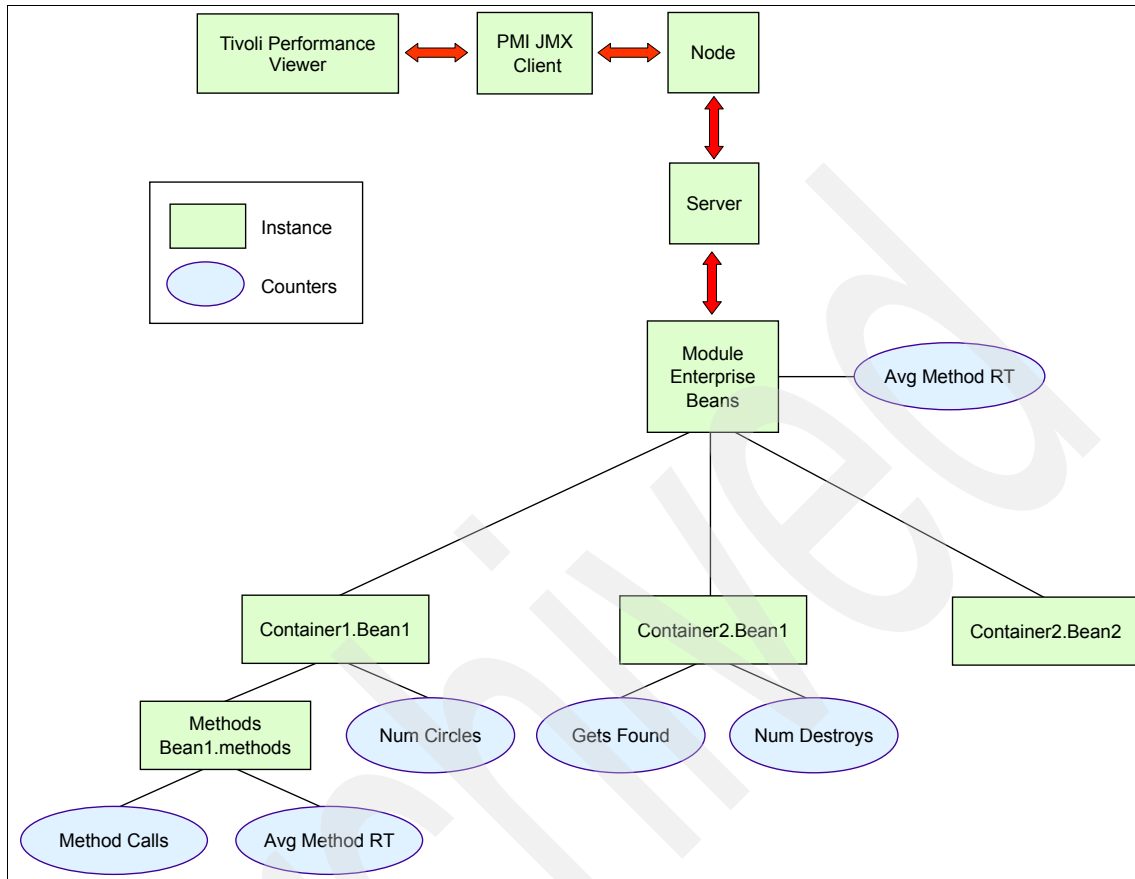


Figure 14-3 Example performance group hierarchy

A subset of counters is available based on the monitoring level chosen for the particular module or instance. Counters are enabled at the module level and can be enabled or disabled for elements within the module. For example, in the figure, if the Module Enterprise Beans module is enabled, its Avg Method RT counter is enabled by default. However, you can then disable the Avg Method RT counter for the Methods Beans1.methods and the aggregate response time reported for the whole Enterprise Beans module will no longer include Methods Bean1.methods data.

Performance data organization

PMI data is provided to clients in a hierarchical structure and organized into modules (resource categories) based on runtime components. Each module has a configuration file in XML format that determines its organization. It specifies unique identifiers for each performance data item per module. A client can use

this unique identifier to fetch the performance data's static and dynamic information.

Performance data is collected for each of the following modules (resource categories). Note that the modules denoted with an asterisk (*) are new since WebSphere Application Server V6:

- ▶ Enterprise Beans

Reports load values, response times, and life cycle activities for EJBs. Examples include the average number of active beans and the number of times bean data is loaded or written to the database. It also reports information about the size and the usage of a cache of bean objects (EJB object pool).

- ▶ JDBC Connection Pools

Reports usage information about connection pools for a database. Examples are the average size of the connection pool, the average number of threads waiting for a connection, the average waiting time in milliseconds for a connection and the average time a connection was in use.

- ▶ JCA Connection Pools

Reports usage information about the J2EE Connector Architecture that enables EJBs to connect and interact with procedural back-end systems such as CICS® and IMS™. Examples are the number of managed connections (physical connections) and the total number of connections (connection handles).

- ▶ JVM Runtime

Reports memory used by a process as reported by the JVM. Examples are the total memory available and the amount of free memory for the JVM. In addition, all performance data that was previously collected in the JVMPI module are now being collected here. Examples are number of garbage collection calls, number of times a thread waits for a lock, and total number of objects allocated in the heap. See 14.2.6, "Using JVMPI facility for PMI statistics" on page 786 for a description of the JVMPI facility.

- ▶ ORB

Reports usage information about the Object Request Broker that enables remote clients to instantiate and look up objects in the application server JVM. Examples are lookup time for a object reference before method dispatch, total number of requests sent to the ORB, and the time it takes for a registered portable interceptor to run.

- ▶ **Servlet Session Manager**

Reports usage information for HTTP sessions. Examples include the total number of sessions being accessed, the average session life time in milliseconds, and the time taken for writing session data to the persistent store.
- ▶ **Thread Pools**

Reports information about the pool of ORB threads that an application server uses to process remote methods and the Web container pools that are used to process HTTP requests coming into the application server. Examples include the average pool size, the number of threads created and destroyed, and the number of concurrently active threads.
- ▶ **Transaction Manager**

Reports transaction information for the container. Examples include the average number of concurrently active transactions (local and global), the average duration of transactions, and the number of transactions committed, rolled back, and timed out.
- ▶ **Web Applications**

Reports load information for the selected Web application and the installed servlets. Examples are the number of loaded servlets, the number of servlet reloads, the total requests that a servlet has processed, and the response time in milliseconds for servlet requests.
- ▶ **Web services Gateway**

Reports usage information from the Web services gateway facility. An example of performance data collected is number of synchronous and asynchronous requests and responses.
- ▶ **System Data**

Reports system level metrics for a node. In WebSphere Application Server (and in the Express edition), this information is available in the application server. In a Network Deployment configuration this information resides in the Node Agent. Examples include CPU utilization and free memory available.
- ▶ **Workload Management (WLM)**

Reports information about enterprise bean workload management. Examples include number of WLM clients serviced, server response time, and number of concurrent requests.
- ▶ **Dynamic Caching**

Reports usage information from the dynamic cache service. Examples include number of client requests, cache misses, and cache hits on disk.

- ▶ **Web services**
Reports information for Web services. Examples include number of loaded Web services, number of requests delivered and processed, request response time, and average size of requests.
- ▶ **Alarm Manager**
Reports information for the Alarm Manager. Examples include the number of alarms cancelled by the application, number of alarms firing per second, or the number of alarms fired.
- ▶ **Object Pool**
Reports information for Object Pools. Examples include the total number of objects created, number of objects requested from the pool, number of objects returned to the pool, and average number of idle object instances in the pool.
- ▶ **Schedulers**
Reports information for the Scheduler service. Examples include the number of tasks that failed to execute, the number of tasks executed successfully, number of tasks executed per second, and many more.
- ▶ **DCS Statistics***
Reports information for the Distribution and Consistence Services (DCS) messages. Examples include the amount of time needed for the synchronization procedure to complete, the number of messages received by the stack, or the incoming message size.
- ▶ **System Integration Bus Service***
Reports information for System Integration Bus service. Examples include the messaging engines and bus communications.
- ▶ **HAManager***
Reports information for high availability. Examples include the number of local groups and the group state rebuild time.

14.2.3 Performance data counters

Each resource category (module) has its own set of performance data counters.

A complete list of all performance data counters for each resource category is included in the WebSphere Application Server V6 InfoCenter article “PMI data organization.” Look at the Sub-topics section at the end of this article. There is a separate table for each resource category. As an example, the ORB counters are shown in Table 14-2 on page 780.

Table 14-2 Counter information for ORB service in PMI

Name	Description	Version	Granularity	Type
reference LookupTime	The time (in milliseconds) to look up an object reference before method dispatch can be carried out	5.0	ORB	Time Statistic
numRequest	The total number of requests sent to the ORB	5.0	ORB	Count Statistic
concurrent Requests	The number of requests that are concurrently processed by the ORB	5.0	ORB	Range Statistic
processing Time	The time (in milliseconds) it takes a registered portable interceptor to run	5.0	per interceptor	Time Statistic

In Table 14-2:

- ▶ Version refers to the version of WebSphere Application Server when the counter was introduced into the PMI framework.
- ▶ Granularity refers to the unit to which data collection is applied for that counter.
- ▶ Type refers to the performance data classification as described in 14.2.1, “Performance data classification” on page 773.

14.2.4 PMI predefined statistic sets

In IBM WebSphere Application Server V6, PMI provides four predefined statistic sets that can be used to enable a set of statistics. These four predefined statistic sets are:

- ▶ None
- ▶ Basic
- ▶ Extended
- ▶ All

You can also use the **Custom** setting to define your own statistic set. Table 14-3 on page 781 provides the details on these options.

Table 14-3 Predefined statistic sets

Statistic set	Description
None	All statistics are disabled.
Basic	Statistics specified in J2EE 1.4, as well as top statistics like CPU usage and live HTTP sessions are enabled. This set is enabled by default and provides basic performance data about runtime and application components.
Extended	Basic set plus key statistics from various WebSphere Application Server components like WLM and Dynamic caching are enabled. This set provides detailed performance data about various runtime and application components.
All	All statistics are enabled.
Custom	Enable or disable statistics individually.

In WebSphere Application Server V5, the statistics were enabled based on a monitoring/instrumentation level. The levels were None, Low, Medium, High and Max {N, L, M, H, X}.

In WebSphere Application Server V6 these instrumentation levels are no longer used. Instead, it introduces a more fine-grained control to enable/disable statistics individually under the Custom statistic set. For example, the counters under the “Enterprise Beans” module can be enabled and disabled as shown in Figure 14-4

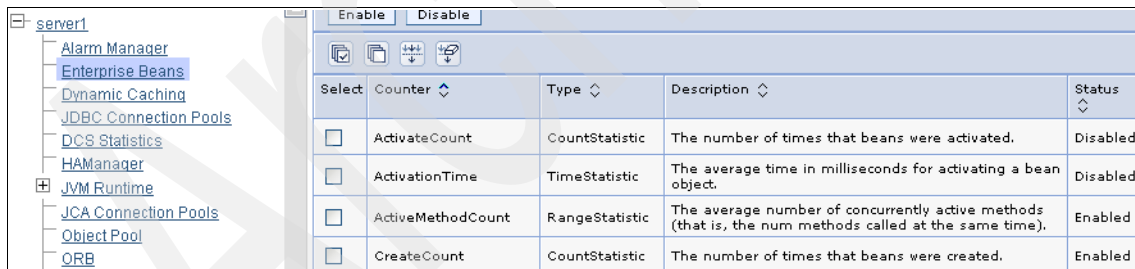


Figure 14-4 Custom statistic set

Important: Data collection can affect performance of the application server. The impact depends on the number and type of counters enabled. Therefore, it is recommended that you enable only those counters that are needed when monitoring a specific scenario.

Sequential counter update

In order to minimize the monitoring overhead, the updates to CountStatistic, AverageStatistic, and TimeStatistic are not synchronized. Since these statistic types track total and average, the extra accuracy is generally not worth the performance cost.

The RangeStatistic and BoundedRangeStatistic are very sensitive, therefore, they are always synchronized.

If needed, updates to all statistic types can be synchronized by checking the **Use sequential counter updates** check box in the PMI General Properties. This is equivalent to setting the instrumentation level to Max in WebSphere Application Server V5.x.

For details on how to configure the Sequential counter update, see 14.2.5, “Enabling the PMI service” on page 782 below.

14.2.5 Enabling the PMI service

In order to monitor a resource with Tivoli Performance Viewer or any JMX client, the PMI service of the application server associated with that resource has to be enabled. The PMI service can be enabled from the Performance Monitoring Service configuration pane in the Administrative Console or by using the **wsadmin** command interface. When running WebSphere Application Server Network Deployment, be sure to enable the PMI service in both the application server and in the Node Agent through the Administrative Console or wsadmin.

Using Administrative Console to enable or change the PMI service for the application server

In WebSphere Application Server V6, PMI is enabled by default - using the Basic statistics set. In order to enable, disable, or change the PMI service settings from the Administrative Console, follow these steps:

1. Click **Servers -> Application servers**.
2. Select your application server (for example Web1) from the list of application servers in the workspace.
3. Select **Performance Monitoring Infrastructure (PMI)**. The configuration panel is shown in Figure 14-5 on page 784.

Note: Alternatively, you can select **Monitoring and Tuning -> Performance Monitoring Infrastructure (PMI) -> <AppServer_Name>** in the Administrative Console. This brings you to the same configuration panel.

4. To enable the PMI service for this application server, select the **Enable Performance Monitoring Infrastructure (PMI)** check box.
5. Optionally, select the check box **Use sequential counter updates** to enable precise statistic updates (see “Sequential counter update” on page 782 for information about this configuration option).
6. Make sure you choose the statistic set that needs to be monitored. The default is Basic.
7. Instead of selecting a predefined statistic set, you can also select **Custom** to selectively enable or disable statistics for fine-grained control. Choose a component from the left side tree. This brings up a table with all counters related to that component on the right hand side of the window. Select the individual statistics that you want to monitor and click **Enable**.

Note: When selecting Custom statistics, there is a Runtime and a Configuration tab available. Only selections done on the Configuration tab survive an application server restart. Changes on the Runtime tab are enabled immediately and can be monitored right away in Tivoli Performance Viewer.

8. Save your configuration.

If PMI was already enabled and you are only changing settings on the Custom -> Runtime tab, then you do not need to restart your application server. If you however enable or disable PMI, change the Use sequential counter updates setting, or change values on the **Custom -> Configuration** tab, then the application server needs to be restarted for the change(s) to take effect.

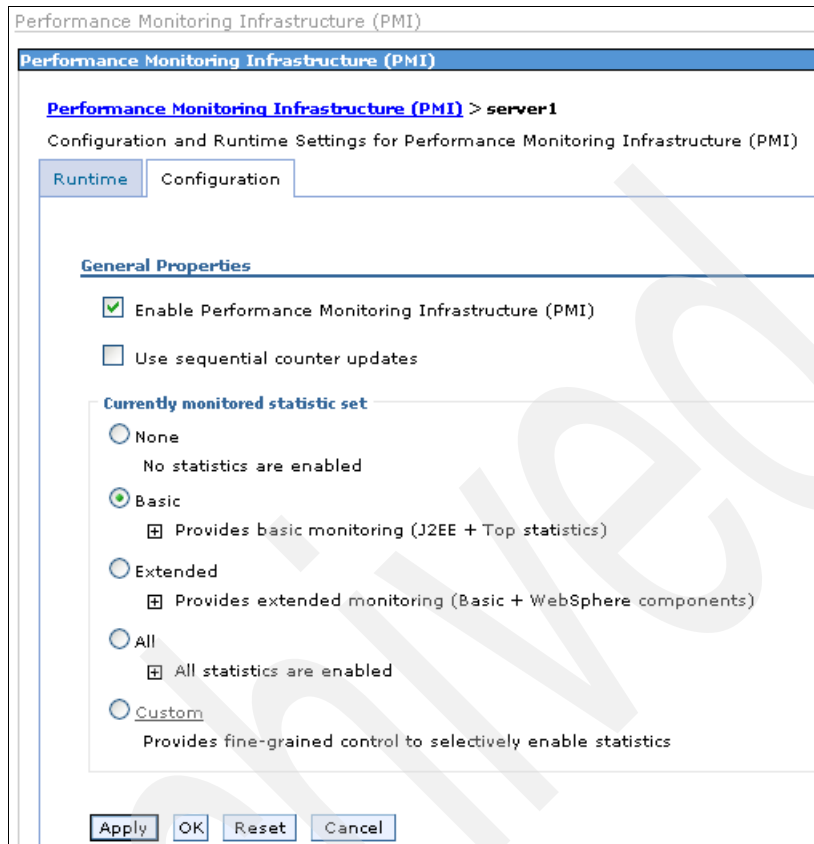


Figure 14-5 Using Administrative Console to enable the PMI service

Using Administrative Console to enable or change the PMI service for the Node Agent

In a Network Deployment environment, you need to enable the PMI service also for the Node Agent(s). In WebSphere Application Server V6, PMI is also enabled by default for the Node Agents. Follow these steps to change the settings:

1. Open the Administrative Console and select **System administration -> Node agents** from the navigation tree.
2. Select **nodeagent**. Make sure you select the Node Agent on the node you wish to monitor.
3. Select **Performance Monitoring Infrastructure (PMI)** from the Additional Properties.

4. Select or deselect the **Enable Performance Monitoring Infrastructure (PMI)** check box.
5. Optionally, select the check box **Use sequential counter updates** to enable precise statistic update.
6. Choose the appropriate statistic set or select **Custom** to enable or disable individual statistics.
7. Click **Apply** or **OK**.
8. Save your configuration changes. Again, a restart of the Node Agent is not needed when changing only settings on the **Custom -> Runtime** tab.

Using wsadmin to enable the PMI service

In order to configure the PMI service of a specific application server, a reference to the PMI service configuration object of that application server is needed. All PMI service configuration objects can be listed using the **wsadmin list PMIService** command.

```
C:\WebSphere\AppServer\bin>wsadmin
WASX7209I: Connected to process "dmgr" on node dmNode using SOAP connector;
The type of process is: DeploymentManager
WASX7029I: For help, enter: "$Help help"
wsadmin>$AdminConfig list PMIService
(cells/dmCell/nodes/app1Node/servers/Advisor1|server.xml#PMIService_11079555
53254)
(cells/dmCell/nodes/app1Node/servers/Ejb1|server.xml#PMIService_110805115791
9)
(cells/dmCell/nodes/app1Node/servers/Web1|server.xml#PMIService_110805072268
3)
(cells/dmCell/nodes/app1Node/servers/nodeagent|server.xml#PMIService_1107435
526812)
(cells/dmCell/nodes/app2Node/servers/Ejb2a|server.xml#PMIService_11080511601
12)
(cells/dmCell/nodes/app2Node/servers/Ejb2b|server.xml#PMIService_11080511618
04)
(cells/dmCell/nodes/app2Node/servers/Web2a|server.xml#PMIService_11080509957
56)
(cells/dmCell/nodes/app2Node/servers/Web2b|server.xml#PMIService_11080509969
37)
(cells/dmCell/nodes/app2Node/servers/nodeagent|server.xml#PMIService_1108047
186422)
(cells/dmCell/nodes/dmNode/servers/dmgr|server.xml#PMIService_1)
wsadmin>
```

Figure 14-6 Using wsadmin to list the PMI service configuration objects

Each line of output contains the PMIService configuration ID that can be used for referencing the PMIService component of a specific application server.

To enable performance data monitoring, use the **wsadmin modify** command with your specific PMI service configuration ID:

```
wsadmin>$AdminConfig modify (cells/dmCell/nodes/app2Node/servers/Ejb2a|
server.xml#PMIService_1108051160112) {{enable true}}
```

Please note that this and all following wsadmin commands need to be written on one line, without any line breaks.

The configuration needs to be saved before restarting the application server. Use the **wsadmin save** command to save the configuration:

```
wsadmin> $AdminConfig save
```

To restart the application server, use these **wsadmin** commands (in a single-server environment, do not specify the node in the **startServer** command):

```
wsadmin>$AdminControl stopServer Ejb2a
WASX7337I: Invoked stop for server "Ejb2a" Waiting for stop completion.
WASX7264I: Stop completed for server "Ejb2a" on node "app2Node"
wsadmin>$AdminControl startServer {Ejb2a} {app2Node}
WASX7262I: Start completed for server "Ejb2a" on node "app2Node"
```

To disable performance data collection, use the **wsadmin modify** command (save the configuration and restart the application server for the change to take effect):

```
wsadmin>$AdminConfig modify (cells/dmCell/nodes/app2Node/servers/Ejb2a|
server.xml#PMIService_1108051160112) {{enable false}}
```

14.2.6 Using JVMPI facility for PMI statistics

The Java Virtual Machine Profiler Interface (JVMPI) is a facility of the JVM used to enable a more comprehensive performance analysis. By enabling the JVMPI interface, the Performance Monitoring Infrastructure can provide more additional performance data such as statistics on garbage collection.

JVMPI is a two-way function call interface between the JVM and an in-process profiler agent. The JVM notifies the profiler agent of various events, such as heap allocations and thread starts. The profiler agent can activate or inactivate specific event notifications, based on the needs of the profiler.

All JVMPI performance data is collected by the JVM module, but JVMPI needs to be enabled for the module to update its counters.

The JVMPI facility is available on the Windows, UNIX, and Linux platforms.

Important: Please be aware that enabling JVMPI introduces a significant overhead on your environment. Therefore, enable JVMPI only when these metrics are needed and do not forget to disable it once you have finished your tests or problem determination.

Performance data provided by JVMPI

Below are the statistics that PMI provides through the use of JVMPI:

- ▶ Garbage collector
 - Number of garbage collection calls
 - Average time in milliseconds between garbage collection calls
 - Average duration in milliseconds of a garbage collection call
- ▶ Monitor
 - Number of times that a thread waits for a lock
 - Average time that a thread waits for a lock
- ▶ Object
 - Number of objects allocated
 - Number of objects freed from heap
 - Number of objects moved in heap
- ▶ Thread
 - Number of threads started
 - Number of threads died

Enabling JVMPI from the Administrative Console

To enable JVMPI reporting for each individual application server or Node Agent, do the following in the WebSphere Administrative Console:

1. Select **Servers** -> **Application servers** or **System administration** -> **Node agents** in the console navigation tree (depending on the JVM you would like to profile).
2. Select the application server or Node Agent from the list of application servers or Node Agents for which JVMPI needs to be enabled.
3. Extend **Java and Process Management** under Server Infrastructure.
4. Click **Process Definition**.
5. Click **Java Virtual Machine** in the Additional Properties.
6. Type `-XrunpmiJvmpiProfiler` into the Generic JVM arguments field as shown in Figure 14-7 on page 788. Add this entry before or after any existing arguments in case you have other arguments already.

Initial Heap Size

Maximum Heap Size

☐ Run HProf

HProf Arguments

☐ Debug Mode

Debug arguments

-Djava.compiler=NONE -Xdeb

Generic JVM arguments

-XrunpmiJvmpiProfiler

Executable JAR file name

☐ Disable JIT

Operating system name

Apply OK Reset Cancel

Figure 14-7 Enabling JVMPI

7. Click **Apply** or **OK**, then **Save** your changes in the WebSphere configuration.
8. Start the application server or Node Agent, or restart the application server or Node Agent if it is currently running.
9. In addition to enabling JVMPI for the application servers and Node Agents, you need to make sure that PMI is enabled with the right settings. Go to **Monitoring and Tuning -> Performance Monitoring Infrastructure (PMI) -> <AppServer_Name> -> Custom**.

For the Node Agents, go to **System administration -> Node agents -> <nodeagent> -> Performance Monitoring Infrastructure (PMI) -> Custom**.

Expand **JVM Runtime** and select the module(s) (Garbage Collection, Object, Thread, Monitor) that you wish to enable counters for. Select the appropriate counters from the table in the right hand pane.

Important: Node Agents and application servers collect data in the JVMPI counters of the JVM module regardless of having the command-line argument specified. Be aware that collected data is not reliable until the `-XrunpmiJvmpiProfiler` argument is specified.

Enabling JVMPI with the command line interface

To enable JVMPI profiling using the **wsadmin** command interface, perform these steps:

1. Start **wsadmin**.
2. Enter the following command at the prompt:

```
wsadmin>$AdminConfig modify (cells/dmCell/nodes/app1Node/servers/Web1|
server.xml#JavaVirtualMachine_1108050722693) {{genericJvmArguments
-XrunpmiJvmpiProfiler}}
```

Tip: To find the available JVMs in your cell, just type `$AdminConfig list JavaVirtualMachine`.

3. Save the configuration (**\$AdminConfig save**) and start the application server or Node Agent, or restart the application server or Node Agent if it was already running for the change to take effect.

Figure 14-8 shows a Tivoli Performance Viewer output with JVMPI enabled. More information about the Tivoli Performance Viewer is found in 14.3, “Using Tivoli Performance Viewer” on page 790.

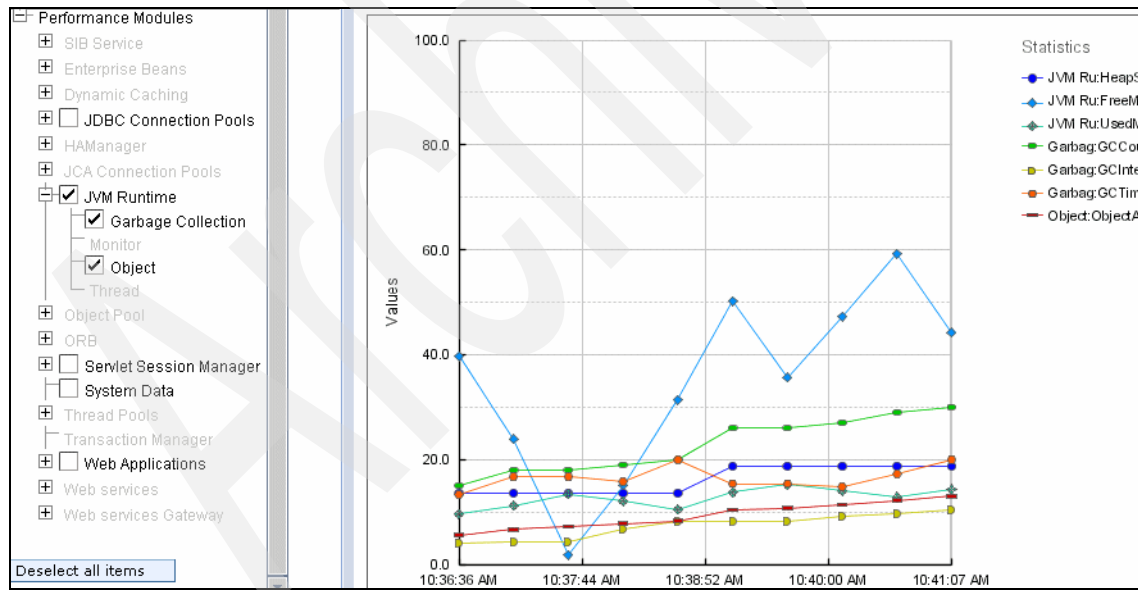


Figure 14-8 Tivoli Performance Viewer: JVM module with JVMPI enabled

Disabling JVMPI profiling

To disable the JVMPI profiling using the Administrative Console:

1. Follow the steps described in “Enabling JVMPI from the Administrative Console” on page 787 to get to the JVM properties for the server or Node Agent.
2. Remove the JVM command line argument `-XrunpmiJvmpiProfiler`.
3. Save your changes and restart your application server or Node Agent.

To disable the JVMPI profiling using wsadmin:

1. Use the **wsadmin** command shown in “Enabling JVMPI with the command line interface” on page 789, but change the `genericJvmArguments` to read:

```
{{genericJvmArguments {}}}
```
2. Save the configuration and restart your application server or Node Agent.

14.2.7 Summary

Performance monitoring is an activity in which you collect and analyze data about the performance of your applications and their environment. We have just discussed the Performance Monitoring Infrastructure that provides the APIs to collect that performance data. This performance data can be monitored and analyzed with:

- ▶ Tivoli Performance Viewer (formerly known as Resource Analyzer), which is included in WebSphere Application Server.
- ▶ Other IBM Tivoli monitoring tools sold separately.
- ▶ User developed monitoring tools.
- ▶ Third party monitoring tools.

14.3 Using Tivoli Performance Viewer

Tivoli Performance Viewer is a real-time monitoring tool that displays PMI data. This tool ships as part of WebSphere Application Server. The tool was formerly known as Resource Analyzer, but was renamed for the WebSphere Application Server V5 release.

In WebSphere Application Server V6, Tivoli Performance Viewer is fully integrated into the Administrative Console, the stand-alone thick Tivoli Performance Viewer Java client is no longer supported.

This tool provides summary reports of key performance data and allows the user to view the data in tabular form or in graphical form. It can also record the information it collects and replay it in a log file.

14.3.1 About Tivoli Performance Viewer

The Tivoli Performance Viewer (TPV) retrieves performance data by periodically polling the PMI service of the application server(s) being monitored. There is a TPV interface running within the Deployment Manager which retrieves data from the TPV engine running within the Node Agents. The TPV engine collects active PMI metrics and transforms the data to display either current value, change of value over previous point, or change of value since buffer reset.

The Node Agents are responsible for querying PMI data from the application servers in the node. In addition, the Node Agents in a Network Deployment environment also hosts a PMI service for monitoring the running state of the physical machine.

To minimize the performance impact, Tivoli Performance Viewer polls the server with the PMI data at an interval set by the user (the default is 30 seconds). The Tivoli Performance Viewer's GUI provides controls that enable you to choose the particular resources and counters to include in the view. There are table and chart views available. You can also store retrieved data in a log file while viewing the data. This log file can later be used for replaying the scenario. The log file can have two different formats, the preferred format is XML.

Figure 14-9 on page 792 shows a high-level overview of how the Tivoli Performance Viewer collects data:

1. The Deployment Manager contacts the Tivoli Performance Viewer Engine (TPV Engine) in the Node Agent to retrieve data.
2. The TPV Engine in the Node Agent pulls PMI data from the application servers on the appropriate node.

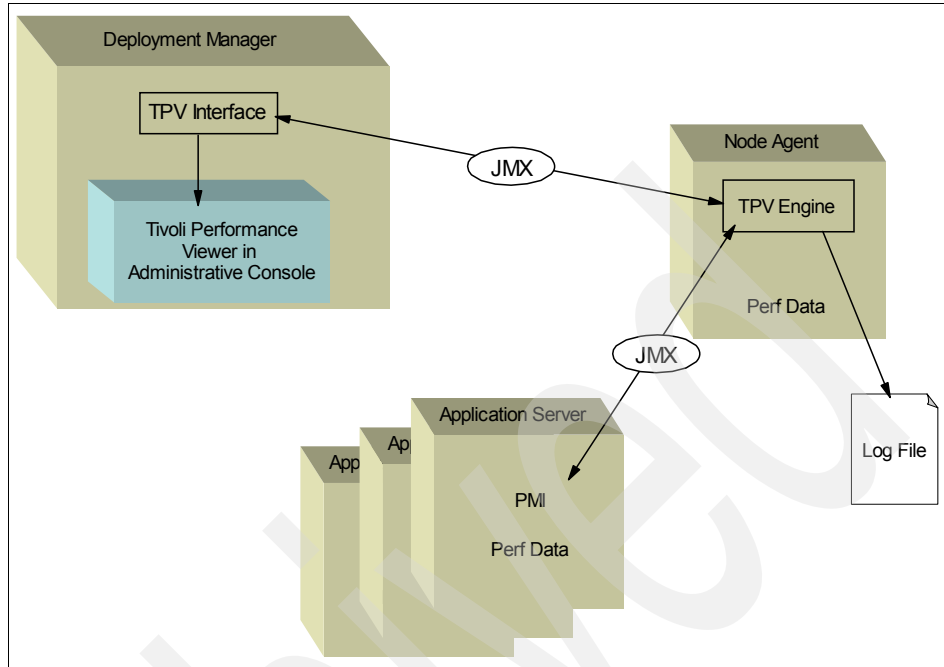


Figure 14-9 Tivoli Performance Viewer infrastructure overview

14.3.2 What can Tivoli Performance Viewer do?

The Tivoli Performance Viewer provides access to a wide range of performance data for three kinds of resources:

- ▶ Application resources (for example, servlet response time and EJB response time).
- ▶ WebSphere runtime resources (for example, application server thread pools and database connection pools).
- ▶ System resources (for example, CPU utilization).

Performance data includes simple counters, statistical data (such as the response time for each method invocation of an enterprise bean), and load data (such as the average size of a database connection pool during a specified time interval). This data is reported for individual resources and aggregated for multiple resources. See 14.2, “Performance Monitoring Infrastructure” on page 771 for more details about performance data organization.

Tivoli Performance Viewer functionality

Depending on which aspects of performance are being measured, you can use the Tivoli Performance Viewer to accomplish the following tasks:

- ▶ View data in real time or view historical data from log files.
- ▶ View data in chart form, allowing comparisons of one or more statistical values for multiple resources on the same chart. In addition, different units of measurement can be scaled to enable meaningful graphic displays.
- ▶ TPV actually allows a user to compare statistical values for multiple resources on the same chart.
- ▶ Record current performance data in a log and replay performance data from previous sessions.
- ▶ Compare data for a single resource to an aggregate (group) of resources on a single node.

Given all this data, the Tivoli Performance Viewer can be used to do the following types of analysis:

- ▶ Monitor real-time performance, such as response times for servlet requests or enterprise bean methods.
- ▶ Detect trends by analyzing logs of data over short periods of time.
- ▶ Determine the efficiency of a configuration of resources (such as the amount of allocated memory, the size of database connection pools, and the size of a cache for enterprise bean objects).
- ▶ Gauge the load on application servers and the average response time for clients.

14.3.3 Starting Tivoli Performance Viewer

To use Tivoli Performance Viewer in WebSphere Application Server V6:

1. Open the Administrative Console, click **Monitoring and Tuning -> Performance Viewer -> Current Activity** in the console navigation tree. The list of application servers and their Collection Status is displayed.
2. Start monitoring the current activity of a server in either of two ways:
 - a. Click the name of the server whose activity you want to monitor.
 - b. Select the check box for the server whose activity you want to monitor, and click **Start Monitoring**. To start monitoring multiple servers at the same time, select the appropriate servers and click **Start Monitoring**.
3. A Tivoli Performance Viewer console panel is displayed, providing a navigation tree on the left and a view of real-time data on the current performance activity of the server on the right.

4. From the navigation tree, select the type of data or server activity that you want to view, the options are listed in Table 14-4.
5. To stop Tivoli Performance Viewer, in the Tivoli Performance Viewer page, check the server that is being monitored, and click **Stop Monitoring**. Or Tivoli Performance Viewer automatically stops monitoring a server when it detects a long period of inactivity.

Table 14-4 Tivoli Performance Viewer options

Option	Description
Advisor	Use the Performance Advisor to examine various data while your application is running. The Performance Advisor provides advice to help tune systems for optimal performance and gives recommendations on inefficient settings by using collected PMI data.
Settings	Configure user and logging settings for Tivoli Performance Viewer. These settings can affect the performance of your application server.
Summary Reports	View summary reports on servlets, enterprise beans (EJBs), EJB methods, connections pools and thread pools in WebSphere Application Server.
Performance Modules	View performance modules that provide graphics and charts of various performance data on system resources such as CPU utilization, on WebSphere pools and queues such as database connection pools, and on customer application data such as servlet response time.

14.3.4 Configuring Tivoli Performance Viewer

The activity monitoring of Tivoli Performance Viewer can be configured on a per-user basis. Any changes made to Tivoli Performance Viewer settings are only for the server being monitored and only affect the user viewing the data.

Configuration for Tivoli Performance Viewer includes user settings and log settings, which directly affect the performance of the application server.

1. User settings

User settings include the refresh rate and buffer size for data collection, also the format options for viewing the data. The detail for these settings are listed in Table 14-5 on page 795

Table 14-5 Tivoli Performance Viewer User settings

Refresh Rate	Specifies how frequently Tivoli Performance Viewer collects performance data for a server from the Performance Monitoring Infrastructure (PMI) service provided by that server. The default is 30 seconds. To collect performance data for the server more frequently, set the refresh rate to a smaller number. To collect performance data less frequently, set the refresh rate to a larger number. The allowed range is 5 to 500 seconds.
Buffer Size	Specifies the amount of data to be stored for a server. Data displayed in Tivoli Performance Viewer is stored in a short in-memory buffer. After the buffer is full, each time a new entry is retrieved the oldest entry is discarded. The default buffer size is 40. Allowed values are 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100. The larger the buffer size, the more memory is consumed. Thus, specify a buffer size that allows you to capture enough monitoring data for analysis without wasting memory by storing unneeded data.
View Data As	<p>Specifies how counter values are displayed. Viewing options include the following:</p> <ul style="list-style-type: none"> ▶ Raw Value Displays the absolute value. If the counter represents load data, such as the average number of connections in a database pool, then Tivoli Performance Viewer displays the current value followed by the average. For example, 18 (avg:5). ▶ Change in Value Displays the change in the current value from the previous value. ▶ Rate of Change Displays the ratio $\text{change}/(T1 - T2)$, where change is the change in the current value from the previous value, T1 is the time when the current value was retrieved, and T2 is the time when the previous value was retrieved.

2. Log settings

The log settings control what happens when **Start Logging** is clicked, for example, in a summary report on the performance of a servlet, enterprise bean (EJB), EJB method, connection pool or thread pool. The details for the log settings are found in Table 14-6 on page 796.

Table 14-6 Tivoli Performance Viewer Log settings

Duration	Specifies the length of time, in minutes, that logging continues, unless Stop Logging is clicked first. (Tivoli Performance Viewer is not intended as a full-time logging solution)
Maximum File Size	Specifies the maximum size, in megabytes, of a single file. Note that Tivoli Performance Viewer automatically zips log files to save space and this parameter controls the pre-zipped file size and not the post-zipped, which is smaller.
Maximum Number of Historical Files	Specifies the number of files Tivoli Performance Viewer writes before stopping. If Tivoli Performance Viewer reaches the maximum file size before the logging duration ends, it continues logging in another file, up to the maximum.
File Name	Specifies the name of the log file. The server name and the time at which the log is started is appended to the log name to help users identifying a log file. The log files are located under the <WAS_INSTALL_ROOT>/profiles/<profile_Name>/logs directory.
Log Output Format	Specifies whether TPV writes log files as XML or in a binary format. Binary format provides a smaller log file when uncompressed.

14.3.5 Tivoli Performance Viewer summary reports

The Tivoli Performance Viewer provides five different summary reports that make important data quickly and easily accessible to help you find performance bottlenecks in your applications and modules.

1. Servlets report

The servlet summary lists all servlets that are running in the current application server. Use the servlet summary view to quickly find the most time intensive servlets and the applications that use them, and to determine which servlets are invoked most often. You can sort the summary table by any of the columns.

Tips:

- ▶ Sort by Avg Response Time to find the slowest servlet or JSP.
- ▶ Sort by Total Requests to find the servlet or JSP used the most.
- ▶ Sort by Total Time to find the most costly servlet or JSP.

2. Enterprise beans report

The Enterprise JavaBeans (EJB) summary lists all enterprise beans running in the server, the amount of time spent in their methods, the number of EJB invocations, and the total time spent in each enterprise bean.

```
total_time = number_of_invocations * time_in_methods
```

Sort the various columns to find the most expensive enterprise bean. Also, if the PMI counters are enabled for individual EJB methods, there is a check box next to the EJB name that you can select to see statistics for each of the methods.

Tips:

- ▶ Sort by Avg Response Time to find the slowest enterprise bean.
- ▶ Sort by Method Calls to find the enterprise bean used the most.
- ▶ Sort by Total Time to find the most costly enterprise bean.

3. EJB Methods report

The EJB method summary shows statistics for each EJB method. Use the EJB method summary to find the most costly methods of your enterprise beans.

Tips:

- ▶ Sort by Avg Response Time to find the slowest EJB method.
- ▶ Sort by Method Calls to find the EJB method used the most.
- ▶ Sort by Total Time to find the most costly EJB method.

4. Connection Pools report

The connection pool summary lists all data source connections that are defined in the application server and shows their usage over time.

Tip: When the application is experiencing normal to heavy usage, the pools used by that application should be nearly fully utilized. Low utilization means that resources are being wasted by maintaining connections or threads that are never used. Consider the order in which work progresses through the various pools. If the resources near the end of the pipeline are underutilized, it might mean that resources near the front are constrained or that more resources than necessary are allocated near the end of the pipeline.

5. Thread pools report

The thread pool summary shows the usage of all thread pools in the application server over time.

Tip: When the application is experiencing normal to heavy usage, the pools used by that application should be nearly fully utilized. Low utilization means that resources are being wasted by maintaining connections or threads that are never used. Consider the order in which work progresses through the various pools. If the resources near the end of the pipeline are underutilized, it might mean that resources near the front are constrained or that more resources than necessary are allocated near the end of the pipeline.

The default monitoring level (Basic) enables all reports except the report on EJB methods. To enable the EJB method summary report, configure PMI service to use a set of statistics that includes EJB method metrics (All or Custom).

14.3.6 Displaying by performance modules

To display performance data by module, extend **Performance Modules** and check the modules to be monitored. Note that the modules need to be enabled in the corresponding PMI service before they can be checked for monitoring.

By default the Performance Module's data is displayed in a graphic chart in the right pane. To view the data as a table, click **View Table**, as shown in Figure 14-10 on page 799. This button changes depending on the view you are in, that is, when you are in the chart view, you can change to the table view and vice versa.

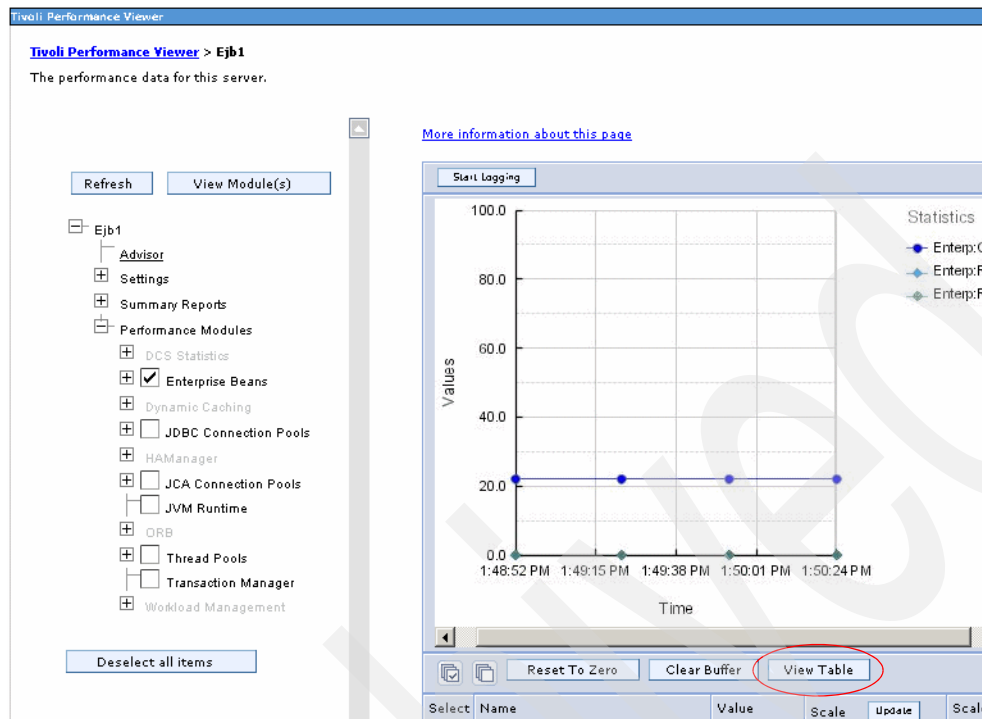


Figure 14-10 Tivoli Performance Viewer - Chart view

Refreshing data

The performance data is refreshed in the following situations:

1. The refresh rate configured in user settings trigger the refreshing.
2. A new performance module is checked in the left pane.

Restriction: Using the GA code of WebSphere V6 we noticed that the automatic refresh works in Internet Explorer but it might not work in other browsers, for example, Mozilla.

3. A new counter is selected in the pane at the bottom.

In WebSphere V6.0.2 the refresh will be non-disruptive so that TPV displays the new data (table and/or graph) without reloading the whole page.

Displaying multiple counters

To analyze the performance data captured by Tivoli Performance Viewer, it is often necessary to view counters from multiple modules. To select multiple counters in one chart or table, scroll down to the bottom pane and check the counters to be monitored as shown in Figure 14-11. The chart or table is refreshed once the counters are selected.

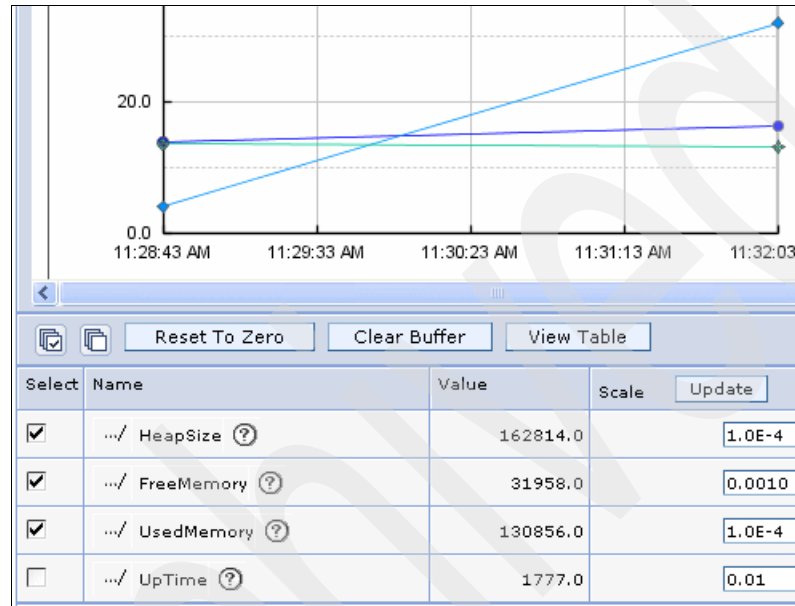


Figure 14-11 Select multiple counters

Recording in a log file

Tivoli Performance Viewer provides an easy way to store real-time data for system resources, WebSphere pools and queues, and applications in log files for later retrieval. You can start and stop logging while viewing current activity for a server, and later replay and analyze the data. The log files are recorded as serialized Java objects or as an XML document.

The steps for recording a log file:

1. Follow the instructions in 14.3.4, “Configuring Tivoli Performance Viewer” on page 794 to set up your logging settings as desired.
2. Click **Start Logging** while viewing summary reports or Performance Modules.
3. When finished, click **Stop Logging**.

Logging stops when the logging duration expires, **Stop Logging** is clicked, or the file size and number limits are reached.

By default, the log files are stored in the <WAS_ROOT>/profiles/<profile_name>/logs/tpv directory on the node on which the server is running. Tivoli Performance Viewer automatically compresses the log file when it finishes to save space. At this point, there must be only a single log file in each .zip file and it must have the same name as the .zip file.

Replaying a log file

The log files can be replayed using the Tivoli Performance Viewer. To replay a log file, do the following:

1. Select **Monitoring and Tuning -> Performance Viewer -> View Logs** in the Administrative Console navigation tree.
2. Select a log file to view using either of the following options:
 - Explicit Path to Log File
Choose a log file from the machine on which the browser is currently running. Use this option if you have created a log file and transferred it to your system. Click **Browse** to search and upload a log file on the local machine.
 - Server File
Specify the path of a log file on the server. In a Network Deployment environment, click the **Browse** button to browse the various nodes and find the log file to view.
3. Click **View Log**, the log is displayed with log control buttons at the top of the view.
4. Adjust the log view as needed. The buttons available for adjusting the view are described in Table 14-7. By default, the data replays at the Refresh Rate specified in the user settings. You can choose one of the Fast Forward modes to play data at a rate faster than the refresh rate.

Table 14-7 Log Viewer buttons

Rewind	Returns to the beginning of the log file.
Stop	Stops the log at its current location.
Play	Begins playing the log from its current location.
Fast Forward	Loads the next data point every three seconds.
Jump Forward	Loads ten data points every three seconds.

5. After a log has been loaded and viewed, return to the View Logs panel to see a list of previously loaded logs. You can select a log for viewing from the list or browse for other logs.

While replaying the log, you can select/deselect Performance Modules in the navigation tree in the left pane or select multiple counters in the bottom pane.

Clearing values from tables and charts

You can clear the values from tables or charts in the Performance Module view at any point by clicking the **Clear Buffer** button at the bottom of the pane. You can then begin populating the table or chart with new data.

Clear Buffer removes PMI data from the table or chart and subsequently only data with a timestamp newer than the time at which the button was clicked is displayed.

Resetting counters to zero

To reset the start time for calculating aggregate data, do the following:

1. Ensure that one or more modules is selected under **Performance Modules** in the Tivoli Performance Viewer navigation tree.
2. Click the **Reset to Zero** button beneath the chart or table.

Some counters report relative values based on how much the value has changed since the counter was enabled. Reset to Zero resets those counters so that they report changes in values since the reset operation. Counters based on absolute values cannot be reset and are not affected by clicking Reset to Zero.

Viewing and modifying chart data

When selected counters are using measurement units that are not proportionally similar, the scaling factor can be set manually to allow a more meaningful display. The following sections explain how you can manually change the scaling factor for the chart view.

Scaling the chart display manually

To manually change the scale of a counter:

1. Place the cursor into the Scale field of the counter selection pane (beneath the graph) for the counter you want to modify.
2. Enter the desired scale value for the counter and click the **Update** button.

Tip: You can enter values for various counters before clicking Update to change the scale value for several counters at once.

The chart view is updated immediately to reflect the change in the scaling factor.

The possible values for the Scale field range from 0 to 100 and show the following relationships:

- < 1 Scaling reduces the value. For example, a scale of 0.5 means that the data points for the variable are reduced to half of their actual values.
- = 1 The value is not scaled. This is the default.
- > 1 Scaling increases the value. For example, a scale of 1.5 means that the data points for the variable are increased to one and one-half times their actual values.

Scaling only applies to the graphed values and has no effect on the data displayed when viewing it in table form.

Refer to 14.3.4, “Configuring Tivoli Performance Viewer” on page 794 for more information about how to configure Tivoli Performance Viewer.

14.3.7 Getting online help

WebSphere Tivoli Performance Viewer provides on-screen links to appropriated sections in the WebSphere InfoCenter on most panels. Simply click **More information about this page**.

You can also go directly to the WebSphere InfoCenter and search for Tivoli Performance Viewer. The InfoCenter is available at:

<http://www.ibm.com/software/webservers/appserv/infocenter.html>

14.4 Other performance monitoring and management solutions

In addition to the tools covered in the previous sections of this chapter, IBM and third-party vendors offer additional performance monitoring and management tools. However, these tools have to be purchased separately.

For example, Tivoli offers IBM Tivoli Monitoring for Web Infrastructure and IBM Tivoli Monitoring for Transaction Performance. For more information, see:

<http://www.ibm.com/software/tivoli/products/monitor-web/>

<http://www.ibm.com/software/tivoli/products/monitor-transaction/>

A basic introduction into Tivoli Monitoring for Transaction Performance can be found in 14.11.1, “Tivoli Monitoring for Transaction Performance V5.3 (TMTP)” on page 831.

Also, several other companies provide performance monitoring, problem determination, and management solutions that can be used with WebSphere Application Server. These products use WebSphere Application Server interfaces, including Performance Monitoring Infrastructure (PMI), Java Management Extensions (JMX), and Request Metrics Application Response Measurement (ARM).

Use the following URL to find a list of IBM Business Partners that offer performance monitoring tools compliant with WebSphere Application Server:

http://www.ibm.com/software/webservers/pw/dhtml/wspperformance/performance_bpsolutions.html

14.5 Developing your own monitoring application

If for some reason you would like to develop your own monitoring application, the PMI offers three interfaces for you:

- ▶ The Java Management Extension (JMX) interface

The JMX interface is accessible through the AdminClient tool. This is the recommended interface for V6.

- ▶ A servlet interface

The servlet interface is perhaps the simplest, requiring minimal programming, as the output is XML.

- ▶ PMI client interface (deprecated)

The PMI client interface is a Java interface that works since Version 3.5.5, it has been deprecated but is still supported in Version 6.0.

Important: Because the PMI client interface has been deprecated, it is not recommended that you start using this interface now for new monitoring applications. Use the JMX interface instead.

If you are already using a self-developed monitoring application that is based on the PMI client API, then it is highly recommended that you rewrite the application to use the JMX interface.

For detailed information about how to implement your own monitoring application using one of these interfaces, refer to the WebSphere Application Server V6 InfoCenter. Search for “Developing your own monitoring applications”, this will list the appropriate articles in the InfoCenter.

14.6 Request Metrics

Request Metrics is a tool that allows you to track individual transactions, recording the processing time in each of the major WebSphere Application Server components. The information tracked may either be saved to log files for later retrieval and analysis, be sent to ARM Agents, or both. Request Metrics is different from PMI instrumentation in that PMI provides the aggregation and averages across all the transactions (such as average servlet response time across transaction A, B, and C), while Request Metrics provides response time for each individual transaction (such as the servlet response time for transaction A.)

As a transaction flows through the system, Request Metrics tracks on additional information so that the log records from each component can be correlated, building up a complete picture of that transaction. The result looks similar to the following:

```
HTTP request /trade/scenario -----> 172 ms
Servlet/trade/scenario -----> 130 ms
    EJB    TradeEJB.getAccountData    --> 38 ms
        JDBC select    --> 7 ms
```

This transaction flow with associated response times can help users target performance problem areas and debug resource constraint problems. For example, the flow can help determine if a transaction is spending most of its time in the Web server plug-in, the Web container, the enterprise bean container or the back-end database. The response time collected for each level includes the time spent at that level and the time spent in the lower levels. In the example above, the response time for the servlet, which is 130 milliseconds, also includes 38 milliseconds from the EJB and JDBC. Therefore, 92 ms can be attributed to the servlet process.

Request Metrics tracks the response time for a desired transaction. For example, tools can inject synthetic transactions. Request Metrics can then track the response time within the WebSphere environment for those transactions. A synthetic transaction is one that is injected into the system by administrators in order to proactively test the performance of the system. This information can help administrators tune the performance of the Web site and take corrective actions should they be needed. Thus, the information provided by Request Metrics might be used as an alert mechanism to detect when the performance of a particular request type goes beyond acceptable thresholds. The filtering mechanism within Request Metrics may be used to focus on the specific synthetic transactions and can help optimize performance in this scenario.

Five types of filters are supported:

- ▶ Source IP filter
- ▶ URI filter
- ▶ EJB method name filter
- ▶ JMS filter
- ▶ Web Services filter

When filtering is enabled, only requests matching the filter generate Request Metrics data, create log records, and/or call the ARM interfaces. This allows work to be injected into a running system (specifically to generate trace information) to evaluate the performance of specific types of requests in the context of normal load, ignoring requests from other sources that might be hitting the system.

14.6.1 Enabling and configuring Request Metrics

Request Metrics are enabled and configured globally for all application servers in the cell. For a single-server environment, the metrics are enabled and configured for all application servers residing on the local node.

The Request Metrics component can be enabled, disabled, and configured at runtime, without having to restart each individual application server in the cell.

To enable the Request Metrics using the Administrative Console, select **Monitoring and Tuning -> Request Metrics** in the navigation tree, specify the following values on the Request Metrics General Properties page (see Figure 14-12 on page 808):

- ▶ Enable Request metrics
Select **Enable Request metrics** to turn on the Request Metrics feature. When it is deselected, the Request Metrics function is disabled.
- ▶ Components to be instrumented
Select the components that are to be instrumented by Request Metrics. The components include All, servlet, enterprise bean, Java Database Connectivity (JDBC), Web services, Java Message Service (JMS), and asynchronous beans. The default is to instrument all components.
- ▶ Trace level
Specify how much trace data is collected for a given transaction. Possible values are:
 - None
No metrics data is collected.

- Hops
Generates instrumentation information on process boundaries. This means, metrics data is collected when a request traverses a process boundary, for example, from one application server JVM to another application server JVM or from the application server to the DB (the JDBC call).
- Performance_debug
In addition to data generated using the Hops level, Performance_debug also generates one additional level of instrumentation data in an application server (for example, from the Web container to the EJB container).
- Debug
Provides detailed instrumentation data for all request levels, including response times for all intra-process servlet and Enterprise JavaBeans (EJB) calls.
- ▶ Standard logs
This setting generates Request Metrics trace record entries in the application server log file (SystemOut.log). However, trace records for HTTP calls (that is requests sent to the Web server) are written to the Web server plug-in log (http_plugin.log).
- ▶ Application Response Measurement (ARM) agent
Optionally, select **Application Response Measurement (ARM) agent** to enable Request Metrics to call an underlying Application Response Measurement (ARM) agent.
To use this feature, you need to install an ARM agent from a provider. Ensure that the native libraries of the ARM implementation are present in the library path, and the ARM API Java archive file is present in the classpath.
- ▶ Agent Type
Here you can specify the ARM agent type. Supported types are ARM4 (any ARM 4.0 compliant agent) and Tivoli_ARM (Tivoli ARM 2.0).
- ▶ ARM transaction factory implementation class name
Specifies the ARM transaction factory implementation class name in the package that is supplied by the provider of the ARM agent.
Enter the name of the ARM transaction factory implementation class name that is present in the ARM library into this field.

Request Metrics

Request Metrics

Request metrics tracks each individual transaction in WebSphere, recording the response time of the components such as time in the webserver or in the Enterprise Java Bean(EJB) container. The information can be saved to log files, sent to Application Response Measurement (ARM) agents, or both. Request Metrics is different from Performance Monitoring Infrastructure (PMI), because PMI provides averages across transactions and includes other data such as system metrics or information on WebSphere threadpool caches

Configuration

General Properties

Additional Properties

☐ Enable Request metrics

Components to be instrumented

ALL

Servlet

EJB

JDBC

* Trace level

Hops

Request Metrics Destination

☐ Standard Logs

☐ Application Response Measurement(ARM) agent

* Agent Type

ARM4

* ARM transaction factory implementation class name

Filters

EJB Filters

URI Filters

Source IP Filters

Web Services Filters

JMS Filters

details

Apply

OK

Reset

Cancel

Figure 14-12 Request Metrics configuration pane

As soon as Request Metrics are enabled and a trace level greater than None is specified and saved to the WebSphere configuration, trace records are written to the System.out JVM log for all application servers for any incoming Web or EJB request.

A detailed description of the trace record format follows in 14.6.2, “Request Metrics trace record format” on page 808.

14.6.2 Request Metrics trace record format

The trace records for Request Metrics data are written to two log files: the Web server plug-in log file (http_plugin.log) and the application server log file

(SystemOut.log). The default directory for SystemOut.log is `<WebSphere_install_root>\profiles\<profile_name>\logs`. The default directory for http_plugin.log is `<WebSphere_install_root>\Plugins\logs\<HTTP_server_name>`. Users might, however, specify these log file names and their locations.

In the Web server plug-in log file, the trace record format is:

```
PLUGIN:
parent:ver=n,ip=n.n.n.n,time=nnnnnnnnnn,pid=nnnn,reqid=nnnnnn,event=nnnn
- current:ver=n,ip=n.n.n.n,time=nnnnnnnnnn,pid=nnnn,reqid=nnnnnn,event=nnnn
type=HTTP detail=some_detail_information elapsed=nnnn bytesIn=nnnn
bytesOut=nnnn
```

In the WebSphere Application Server log file, the trace record format is:

```
PMRM0003I:
parent:ver=n,ip=n.n.n.n,time=nnnnnnnnnn,pid=nnnn,reqid=nnnnnn,event=nnnn
- current:ver=n,ip=n.n.n.n,time=nnnnnnnnnn,pid=nnnn,reqid=nnnnnn,event=nnnn
type=TTT detail=some_detail_information elapsed=nnnn
```

The trace record format is composed of two correlators: a parent correlator and current correlator. The parent correlator represents the upstream request and the current correlator represents the current operation. If the parent and current correlators are the same, then the record represents an operation that occurred as it entered WebSphere Application Server.

To correlate trace records for a particular request, collect records with a message ID of PMRM0003I from the appropriate application server log files and the PLUGIN trace record from the Web server plug-in log file. Records are correlated by matching current correlators to parent correlators. The logical tree can be created by connecting the current correlators of parent trace records to the parent correlators of child records. This tree shows the progression of the request across the server cluster. Refer to the article “Why use request metrics?” in the InfoCenter for an example of the transaction flow.

The parent correlator is denoted by the comma separating fields following the keyword "parent:". Likewise, the current correlator is denoted by the comma separating fields following "current:".

The fields of both parent and current correlators are listed in Table 14-8 on page 810:

Table 14-8 Request Metrics: Parent and current correlators

Field	Description
ver	The version of the correlator. For convenience, it is duplicated in both the parent and current correlators.
ip	The IP address of the node of the application server that generated the correlator.
pid	The process ID of the application server that generated the correlator.
time	The start time of the application server process that generated the correlator.
reqid	An ID assigned to the request by Request Metrics, unique to the application server process.
event	An event ID assigned to differentiate the actual trace events.

Following the parent and current correlators, is the metrics data for timed operation (see Table 14-9):

Table 14-9 Request Metrics: Metrics data for timed operation

Field	Description
type	A code representing the type of operation being timed. Supported types include URI, EJB, JDBC, WebServices, JMS, and AsyncBeans. (See the description of each of these types in Table 14-10 below.)
detail	Identifies the name of the operation being timed.
elapsed	The measured elapsed time in <units> for this operation, which includes all sub-operations called by this operation. The unit of elapsed time is milliseconds.
bytesIn	The number of bytes from the request received by the Web server plug-in.
bytesOut	The number of bytes from the reply sent from the Web server plug-in to the client.

The type and detail fields are described in Table 14-10 on page 811.

Table 14-10 Request Metrics: Type and data fields

Field	Description
HTTP	The Web server plug-in generates the trace record. The detail is the name of the URI used to invoke the request.
URI	The trace record was generated by a Web component. The URI is the name of the URI used to invoke the request.
EJB	The fully qualified package and method name of the enterprise bean.
JDBC	The values select, update, insert or delete for prepared statements. For non-prepared statements, the full statement can appear.
Web Services Provider	<p>The trace record was generated at the Web Services server side. The details include the WSDL port name, operation name, transport name, target name space, and input message name in the following format:</p> <pre>wsprovider:<wsdlPortName>.<wsdlOperationName>? transport=<transportName>&namespace=<targetName space>&input=<inputMessageName></pre>
Web Services Requestor	<p>The trace record was generated at the Web Services client side. The details include the WSDL port name, operation name, transport name, and parameter name list in the following format:</p> <pre>wsrequestor:<wsdlPortName>.<wsdlOperationName> ?transport=<transportName>&parameters=<parameterNameList></pre>
JMS	The trace record was generated by a JMS MDB. The details include the message destination name, topic name if applicable, and method selector if applicable.
JMS_produceMessage	The trace record was generated when sending a JMS message. The details include the message destination (queue or topic space) name and topic name if applicable.
JMS_readMessage	The trace record was generated when synchronously reading a JMS message. The details include the message destination (queue or topic space) name and topic name if applicable.

Field	Description
COMMONJ_WORK_POOLED	The trace record was generated in AsyncBeans work. The detail is the class name for the work.
COMMONJ_WORK_TIMER	The trace record was generated in AsyncBeans alarm. The detail is the class name for the alarm.

Examples of URI and EJB Request Metrics records are shown in Example 14-1 and Example 14-2 on page 812.

Example 14-1 URI Request Metrics

```
[11/29/03 9:46:31:578 EST] 7bca033b PmiRmArmWrapp I PMRM0003I:
parent:ver=1,ip=192.168.0.1,time=1016556185102,pid=884,reqid=4096,event=1
- current:ver=1,ip=192.168.0.1,time=1016556186102,pid=884,reqid=4096,event=1
type=URI detail=/hitcount elapsed=60
```

Example 14-2 EJB (EJB is the child of the servlet request in the downstream process)

```
[11/29/03 9:00:16:812 EST] 6d23ff00 PmiRmArmWrapp I PMRM0003I:
parent:ver=1,ip=192.168.0.1,time=1016556186102,pid=884,reqid=4096,event=1
- current:ver=1,ip=192.168.0.2,time=1016556122505,pid=9321,reqid=8193,event=1
type=EJB detail=com.ibm.defaultapplication.IncrementBean.increment elapsed=10
```

14.6.3 Filters

Filters are important for producing only the output necessary for a given monitoring task. A named URI, EJB, or client IP address can be specified and only the needed metrics data is produced. The performance load added to the containers by the Request Metrics component is reduced by adding more restrictive filters.

For HTTP requests arriving at a Web container, it is possible to filter on the URI and client IP address. For incoming requests to the EJB container, it is possible to filter on the bean method name. All filters are listed and described here:

- ▶ **Source IP address filters:** Requests are filtered based on a known IP address. You can specify a mask for an IP address using the asterisk (*). If used, the asterisk must always be the last character of the mask, for example 127.0.0.*, 127.0.*, 127*.
- ▶ **URI filters:** Requests are filtered based on the URI of the incoming HTTP request. The rules for pattern matching are the same as for matching source IP address filters.
- ▶ **Enterprise bean method name filters:** Requests are filtered based on the full name of the enterprise bean method. As with IP address and URI filters,

you can use the asterisk (*) to provide a mask. The asterisk must always be the last character of a filter pattern.

- ▶ **JMS filters:** Requests are filtered based on the queue destination names and topic destination names. You can specify the value as `destination=aaa:topic=bbb`. The asterisk (*) can be specified as the last character of each filter segment.
- ▶ **Web Services filters:** Requests are filtered based on the wsdl port, operation and namespace. You can specify the value as `wsdlPort=aaa:op=bbb:nameSpace=ccc`. The asterisk (*) can be specified as the last character of each filter segment.
- ▶ **Filter combinations:** If both URI and source IP address filters are active, then Request Metrics require a match for both filter types. If neither is active, all requests are considered a match.

It is important to understand that filters are applied to requests as they enter the application server from the client (at the Web or EJB container level). Depending on defined filters, the request is either marked to have metrics generated or not marked to mean that no Request Metrics records should be produced for the duration of the request.

14.7 Performance Advisors

Understanding how to tune WebSphere applications is often difficult. With IBM WebSphere Application Server V5.0.2, V5.1 and V6.0, the Performance Advisors come to the rescue. There are two Performance Advisors that provide suggestions to help tune systems for optimal performance. Both advisors use the Performance Monitoring Infrastructure (PMI) data to suggest configuration changes to the WebSphere thread pools, connection pools, prepared statement cache, session cache, heap size, etc.

The two Performance Advisors are:

- ▶ Runtime Performance Advisor
- ▶ Performance Advisor in Tivoli Performance Viewer (TPV Advisor)

The first advisor, the Runtime Performance Advisor, which executes in the application server process, is configured through the WebSphere Administrative Console. Running in the application server's JVM, this advisor periodically checks for inefficient settings, and issues recommendations as standard WebSphere warning messages. These recommendations are displayed both as warnings in the Administrative Console under WebSphere Runtime Messages and as text in the SystemOut.log file. Enabling the Runtime Performance Advisor has minimal system performance impact.

The second, the TPV Advisor, runs in the Tivoli Performance Viewer (TPV) and is also configured through the WebSphere Administrative Console. It provides recommendations on inefficient settings at a specific point in time. The TPV Advisor provides additional advice to the Runtime Performance Advisor. For example, TPV also provides advice on setting the dynamic cache size, setting the JVM heap size, and using the DB2 Performance Configuration Wizard.

Table 14-11 gives a summary of the two Performance Advisors.

Table 14-11 Performance Advisors summary

	Runtime Performance Advisor	Performance Advisor in Tivoli Performance Viewer (TPV)
Location of execution	Application server	TPV in the Administrative Console
Location of tool	Administrative Console	TPV
Output	SystemOut.log file and WebSphere run-time messages in the Administrative Console	TPV in the Administrative Console
Frequency of operation	Every 10 seconds in background	When you select to refresh in TPV
Types of advice	<ul style="list-style-type: none"> ▶ ORB service thread pools ▶ Web container thread pools ▶ Connection pool size ▶ Persisted session size and time ▶ Prepared statement cache size ▶ Session cache size 	<ul style="list-style-type: none"> ▶ ORB service thread pools ▶ Web container thread pools ▶ Connection pool size ▶ Persisted session size and time ▶ Prepared statement cache size ▶ Session cache size ▶ Dynamic cache size ▶ JVM heap size ▶ DB2 Performance Configuration Wizard

Note: It is planned to add a new type of advice for the Runtime Performance Advisor in WebSphere V6.0.2. This new advice will provide lightweight memory leak detection.

Figure 14-13 on page 815 shows the simplified architecture of the Performance Advisors.

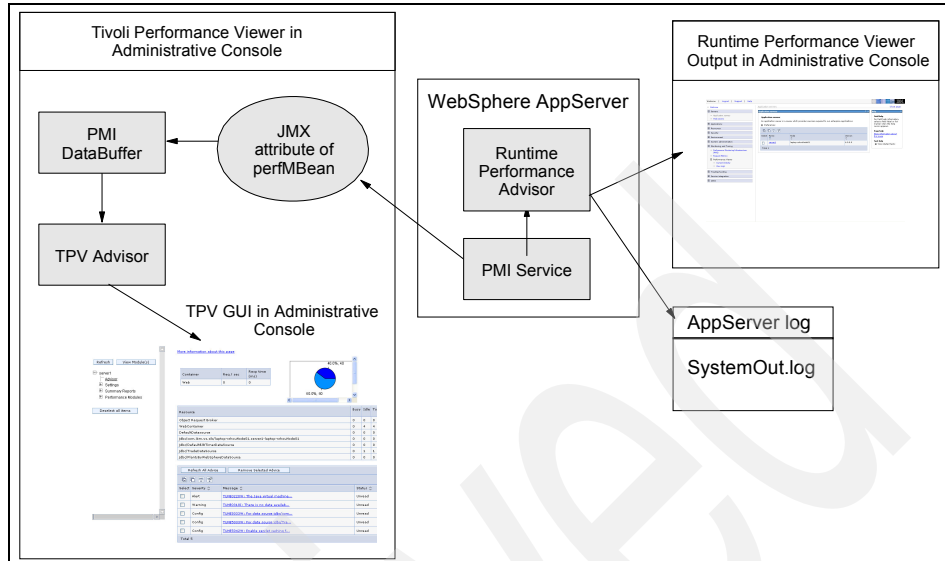


Figure 14-13 Simplified Performance Advisors architecture

For more information about PMI and TPV, refer to 14.2, “Performance Monitoring Infrastructure” on page 771 and 14.3, “Using Tivoli Performance Viewer” on page 790.

14.7.1 Runtime Performance Advisor configuration settings

This section lists the various settings within the Administrative Console under the Runtime Performance Advisor Configuration panel.

To view the Runtime Performance Advisor configuration page, click **Servers -> Application servers -> <AppServer_Name> -> Runtime Performance Advisor Configuration**.

Configuration tab

- Enable Runtime Performance Advisor

Specifies whether the Runtime Performance Advisor runs.

The Runtime Performance Advisor requires that the Performance Monitoring Infrastructure (PMI) be enabled but it does not require that individual counters are enabled. When a counter that is needed by the Runtime Performance Advisor is not enabled, the Runtime Performance Advisor enables it automatically. When disabling the Runtime Performance Advisor, you might want to also disable PMI or the counters that Runtime Performance Advisor

enabled. The following counters might be enabled by the Runtime Performance Advisor:

- **Thread Pools module:** Pool Size and Active Threads
 - **JDBC Connection Pools module:** Pool Size, Percent used, Prepared Statement Discards
 - **Servlet Session Manager module:** External Read Size, External Write Size. External Read Time, External Write Time, No Room For New Session
 - **System Data module:** CPU Utilization and Free Memory
 - **JVM module:** Free memory and Total memory
- Calculation Interval
- PMI data is taken over an interval of time and averaged to provide advice. The interval specifies the length of time over which data is taken for this advice. Therefore, details within the advice messages will appear as averages over this interval.
- Maximum warning sequence
- The maximum warning sequence refers to the number of consecutive warnings issued before the threshold is updated. For example, if the maximum warning sequence is set to 3, then the advisor only sends three warnings to indicate that the prepared statement cache is overflowing. After that, a new alert is only issued if the rate of discards exceeds the new threshold setting.
- Number of processors
- Specifies the number of processors on the server.

Runtime tab

The four configuration settings from the Configuration tab are also available on the Runtime tab.

In addition, there is a Restart button on this tab. Selecting **Restart** reinitializes the Runtime Performance Advisor with the last information saved to disk.

Note that this action also resets the state of the Runtime Performance Advisor. For example, the current warning count is reset to zero for each message.

14.7.2 Advice configuration settings

WebSphere Application Server also allows you to enable and disable advice in the Advice configuration panel. Some advice applies only to certain configurations, and can only be enabled for those configurations. For example,

Unbounded ORB Service Thread Pool Advice is only relevant when the ORB Service thread pool is unbounded, and can only be enabled when the ORB thread pool is unbounded.

To view this configuration page, click **Servers -> Application servers -> <AppServer_Name> -> Runtime Performance Advisor Configuration -> Advice configuration**. Below is a description of the information found on this panel.

Configuration tab

- ▶ Advice name
Specifies the advice that you can enable or disable.
- ▶ Advice applied to component
Specifies the WebSphere Application Server component to which the runtime performance advice applies, for example, Web Container or Data Source.
- ▶ Advice status
Specifies whether the advice is stopped or started.

There are only two values - Started and Stopped. Started means that the advice runs if the advice applies. Stopped means that the advice does not run.

Runtime tab

- ▶ Advice name and Advice applied to component are identical to the Configuration tab.
- ▶ Advice status
On the Runtime tab, the advice status has one of three values - Started, Stopped or Unavailable. Started means that the advice is being applied. Stopped means that the advice is not applied. Unavailable means that the advice does not apply to the current configuration (such as Persisted Session Size advice in a configuration without persistent sessions).

14.7.3 Using the Runtime Performance Advisor

In order to obtain advice, you must first enable the performance monitoring service through the Administrative Console and restart the server. If running Network Deployment, you must enable the PMI service on both the application servers and on the Node Agent and restart the servers and Node Agent. In WebSphere Application Server V6, PMI is enabled by default on the servers and on the Node Agent(s).

The Runtime Performance Advisor enables the appropriate monitoring counter levels for all enabled advice. If there are specific counters that are not wanted, disable the corresponding advice in the Runtime Performance Advisor panel, and disable unwanted counters. See 14.2.5, “Enabling the PMI service” on page 782 for details. Then, do the following:

1. In the WebSphere Administrative Console select **Servers -> Application servers**.
2. Click **<AppServer_Name> -> Runtime Performance Advisor Configuration**.
3. On the Configuration tab select the **Number of processors**.
Selecting the appropriate settings for the system's configuration ensures accurate performance advice.
4. Optionally, select the **Calculation Interval** and the **Maximum warning sequence**.
5. Click **Apply** and **Save** your changes.
6. Select the **Runtime** tab and click **Restart**.
7. Simulate a production level load.

For load testing tools and how to perform the load testing, see 17.3, “Tools of the trade” on page 945.

If you are using the Runtime Performance Advisor in a test environment, or doing any other tuning for performance, simulate a realistic production load for your application. The application should run this load without errors. This simulation includes the number of concurrent users typical for peak periods, and drives system resources, such as CPU and memory to the levels expected during peak production periods. The Performance Advisor can provide some type of advice only when the CPU utilization exceeds a sufficiently high level. For example, advice on thread pool size is only issued if there is a problem with it (which can usually only be detected when the system is under load). Other types of advice, however, are always issued, for example advice related to the Prepared statement cache or the Servlet Session module.

8. Once a stable production level load is reached, select the check box to **Enable** the Runtime Performance Advisor. This way, you will achieve the best results for performance tuning. Click **OK**.
9. Select **Warning** in the Administrative Console under **Troubleshooting -> Runtime Messages** or look at the SystemOut.log file, located in the **<install_root\logs\server_name>** directory to view tuning advice. Some messages are not issued immediately.
10. Change your application server configuration based on the received advice.

Important: As with any analysis and tuning, make sure that only one parameter is changed, and then the results monitored. This provides an effective method of backing out of the change if it proves detrimental to the environment. Also, making multiple changes could result in undesired results, because many options are dependent on other settings.

Although the Performance Advisors attempt to distinguish between loaded and idle conditions, misleading advice might be issued if the advisor is enabled while the system is ramping up or down. This result is especially likely when running short tests. Although the advice helps in most configurations, there might be situations where the advice hinders performance. Due to these conditions, advice is not guaranteed. Therefore, test the environment with the updated configuration to ensure it functions and performs as expected.

Tips for using the Runtime Performance Advisor:

- ▶ Enable PMI in the application server *and* in the Node Agent if running Network Deployment.
- ▶ Be sure to configure the correct number of processors.
- ▶ Simulate a production level load.
 - Ensure the application runs without exceptions/errors.
 - Runtime Performance Advisor only provides advice when CPU utilization is high.
- ▶ Once production load level is reached, enable the Runtime Performance Advisor.
- ▶ Apply advice, restart the application server, and re-test.
- ▶ More details can be found in the WebSphere InfoCenter section “Using the Runtime Performance Advisor”.

14.7.4 Runtime Performance Advisor output

After completing the configuration steps, the Advisor will then begin to report recommendations into the SystemOut.log. Example 14-3 shows sample output from the Advisor.

Example 14-3 Sample output from the Runtime Advisor

```
[6/11/04 15:20:42:484 EDT] 6a1d6f88 TraceResponse W TUNE0208W: Data source
jdbc/TradeDataSource does not seem to be in use. If the data source is used
occasionally, then decrease the number of connections in the pool, by setting
```

minConnections to 0, and maxConnections to 3. If the data source is never used, then delete the data source.

Additional explanatory data follows.

Pool utilization: 0%.

This alert has been issued 1 time(s) in a row. The threshold will be updated to reduce the overhead of the analysis.

[6/11/04 15:24:54:953 EDT] 6a1d6f88 TraceResponse W TUNE0214W: The session cache for Trade3#trade3Web.war is smaller than the average number of live sessions. Increasing the session cache size to at least 1,300 may improve performance.

Additional explanatory data follows.

Session cache size (the maximum in memory session count): 1,000.

Current live sessions: 1,300.

Average live sessions over the last sampling interval: 1,300.

This alert has been issued 1 time(s) in a row. The threshold will be updated to reduce the overhead of the analysis.

In addition, the Runtime Advisor recommendations can also be displayed in the Administrative Console through the WebSphere Runtime Messages. The Runtime Advisor messages are displayed as warnings. Figure 14-14 shows a sample of the output from the advisor.

Runtime Events		
Runtime events propagating from the server		
Preferences		
		
Timestamp	Message Originator	Message
Oct 19, 2004 9:28:35 AM EDT	com.ibm.ws.performance.tuning.serverAlert.TraceResponse	TUNE0208W: Data source jdbc/PlantsByWebSphereDataS
Oct 19, 2004 9:28:35 AM EDT	com.ibm.ws.performance.tuning.serverAlert.TraceResponse	TUNE0206W: Decreasing the connection pool for data
Oct 19, 2004 9:28:35 AM EDT	com.ibm.ws.performance.tuning.serverAlert.TraceResponse	TUNE0208W: Data source jdbc/DefaultEJBTimerDataSou
Oct 19, 2004 9:28:35 AM EDT	com.ibm.ws.performance.tuning.serverAlert.TraceResponse	TUNE0208W: Data source jdbc/com.ibm.ws.sib/laptop-
Oct 19, 2004 9:28:35 AM EDT	com.ibm.ws.performance.tuning.serverAlert.TraceResponse	TUNE0208W: Data source DefaultDatasource does not
Total 5		

Figure 14-14 Runtime Events display of Runtime Advisor output

Click the link for more details on the message; Figure 14-15 on page 821 is an example of such a message.

[Runtime Events](#) > **Message Details**

Runtime events propagating from the server

General Properties

Message
TUNE0206W: Decreasing the connection pool for data source jdbc/TradeDataSource by setting the minimum size to 0 and the maximum size to 3 may improve performance. Additional explanatory data follows. Pool utilization: 1.2%. This alert has been issued 1 time(s) in a row. The threshold will be updated to reduce the overhead of the analysis.

Message type
Warning

Explanation
Decreasing the size supports better pooling and frees memory resources.

User action
From the administrative console, click: Resources > JDBC Providers > JDBC_provider > Data Sources > data_source > Connection pool properties.

Message Originator
com.ibm.ws.performance.tuning.serverAlert.TraceResponse

Source object type
RasLoggingService

Timestamp
Oct 19, 2004 9:28:35 AM EDT

Thread Id
15

Node name
laptop-rzhouNode01

Server name
server1

[Back](#)

Figure 14-15 Detail of Runtime Advisor message

As mentioned in 14.7.3, “Using the Runtime Performance Advisor” on page 817, use these recommendations as input to change your configuration, restart the application server(s), and re-test.

14.7.5 Using TPV Advisor

The Performance Advisor in Tivoli Performance Viewer (TPV) provides advice to help tune systems for optimal performance and gives recommendations on inefficient settings by using collected Performance Monitoring Infrastructure (PMI) data. Advice is obtained by selecting the **Advisor** link in the TPV navigation tree. See Figure 14-16 on page 822.

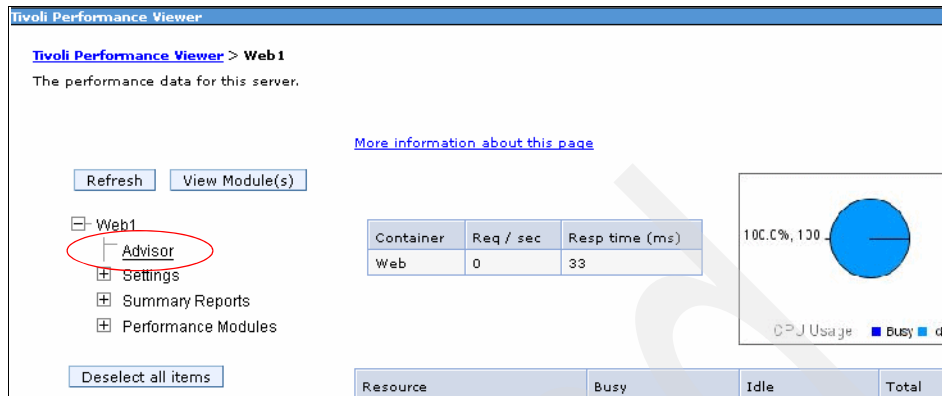


Figure 14-16 Select Advisor in TPV

The Performance Advisor in TPV provides more extensive advice than the Runtime Performance Advisor. For example, TPV provides also advice on setting the dynamic cache size, setting the JVM heap size, and using the DB2 Performance Configuration Wizard.

In order to obtain advice, follow these steps:

1. Enable PMI data collection

Enable PMI data collection (as explained in “Enabling the PMI service” on page 782) and select the **Extended** statistic set. The statistic set (also called the monitoring level) determines which data counters are enabled. You can select the statistics set dynamically on the Runtime tab, without restarting the server. The monitoring level directly influences the type of advice you obtain.

The Performance Advisor in TPV normally uses the Extended monitoring level but it can also use a few of the more expensive counters (to provide additional advice) and provide advice on which counters can be enabled. For example, advice for session size needs the PMI statistic set to All. Or, you can use the PMI Custom monitoring level to enable the Servlet Session Manager SessionObjectSize counter. The monitoring of the SessionSize PMI counter is expensive and is therefore not in the Extended PMI statistic set.

2. In the Administrative Console, click **Monitoring and Tuning -> Performance Viewer -> Current Activity**.

3. Simulate a production level load

Simulate a realistic production load for your application, if you use the Performance Advisor in a test environment, or do any other performance tuning. The application must run this load without errors. This simulation includes numbers of concurrent users typical of peak periods, and drives system resources, for example, CPU and memory to the levels that are

expected during peak production periods. The Performance Advisor can provide some type of advice only when the CPU utilization exceeds a sufficiently high level. For example, advice on thread pool size is only issued if there is a problem with it (which can usually only be detected when the system is under load). Other types of advice, however, are always issued, for example, advice related to the Prepared statement cache or the Servlet Session module.

4. Start logging for Tivoli Performance Viewer (refer to “Recording in a log file” on page 800).
5. Click **Advisor** in the Tivoli Performance Viewer navigation tree. Tuning advice appears in the advice table shown in Figure 14-17. Click **Refresh All Advice** on top of the advice table to recalculate the advice based on the current data in the buffer. Since PMI data is taken over an interval of time and averaged to provide advice, details within the advice message appear as averages.


Refresh All Advice			
Remove Selected Advice			
			
Select	Severity	Message	Status
<input type="checkbox"/>	Warning	TUNE0324W: There was an unknown er...	Unread
<input type="checkbox"/>	Warning	TUNE0303W: Number of threads workin...	Unread
<input type="checkbox"/>	Warning	TUNE0324W: There was an unknown er...	Unread
<input type="checkbox"/>	Alert	TUNE0220W: The Java virtual machine...	Read
<input type="checkbox"/>	Warning	TUNE0318I: There is no data availab...	Unread
Page: 1 of 2 Total 8			

Figure 14-17 Tivoli Performance Viewer Advisor table

6. Change your application server configuration based on the advice. Because Tivoli Performance Viewer refreshes advice at a single instant in time, take the advice from the peak load time.

As mentioned before, the Performance Advisors attempt to distinguish between loaded and idle conditions, misleading advice might be issued if the advisor is enabled while the system is ramping up or down. This result is especially likely when running short tests. Although the advice helps in most configurations, there might be situations where the advice hinders performance. Because of these conditions, advice is not guaranteed. Therefore, test the environment with the updated configuration to ensure it functions and performs well.

Over a period of time the advisor might issue differing advice. The differing advice is due to load fluctuations and run-time state. When differing advice is received, you need to look at all advice and the time period over which they

were issued. You must take advice during the time that most closely represents the peak production load.

Performance tuning is an iterative process. After applying advice, simulate a production load, update the server configuration based on the advice, and retest for improved performance. This procedure is continued until optimal performance is achieved.

Important: As we already stated, make sure that only one parameter is changed and then the results monitored. This provides an effective method of backing out of the change if it proves detrimental to the environment. Also, making multiple changes could result in undesired results, because many options are dependent on other settings.

14.7.6 TPV Advisor output

As the TPV Advisor executes, output is displayed in the Tivoli Performance Viewer panel. Figure 14-18 on page 825 shows an example of this output.

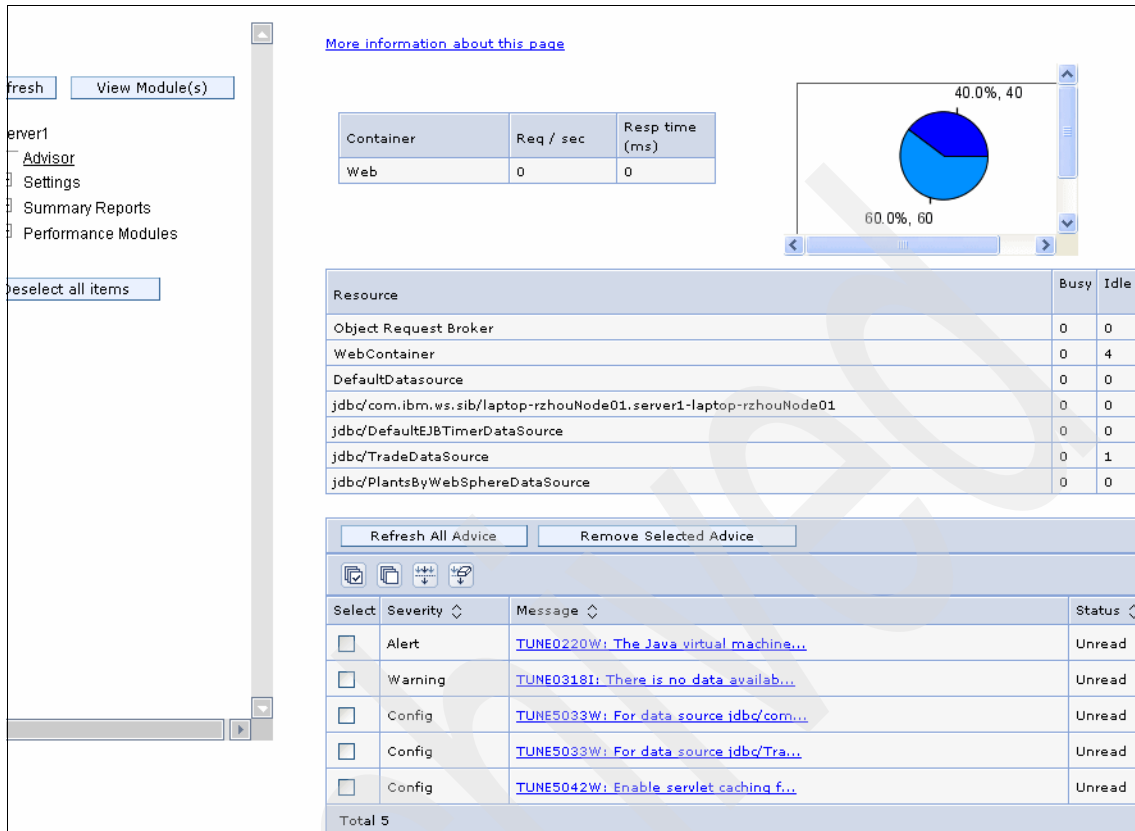


Figure 14-18 TPV Advisor output

To see the details of a specific advisor message, click the message in the advisor table. You can view the details of the message including the Message, Severity, Description, User Action and Detail. Figure 14-19 on page 826 provides an example.

Message
TUNE0220W: The Java virtual machine is spending a considerable amount of time in garbage collection. Consider increasing the heap size.
Severity
Alert
Description
There are many reasons why an application may spend too much time in garbage collection. Often, it indicates an application with many short-lived objects. However, it can also indicate a heap that is too small, causing the JVM's memory manager to thrash. Increasing the heap size will help in this case, by lengthening the time between garbage collection calls, but will also increase the time needed for each garbage collection.
User Action
From the administrative console, click: Application Servers > Server > Java and Process Management > Process Definition > Java Virtual Machine.
Detail
Percentage of time spent in garbage collection: 50%
Back

Figure 14-19 Tivoli Performance Viewer Advisor detail message

14.8 Dynamic Cache Monitor

J2EE applications have high read/write ratios and can tolerate small degrees of latency in the currency of their data. WebSphere Application Server consolidates several caching activities, including servlets, Web services, and WebSphere commands into one service called the *dynamic cache*. These caching activities work together to improve application performance, and share many configuration parameters, which are set in an application server's dynamic cache service.

Therefore, the dynamic cache opens a field for significant gains in server response time, throughput, and scalability. You can use the dynamic cache to improve the performance of servlet and JSP files by serving requests from an in-memory cache. Cache entries contain servlet output, results of servlet execution, and metadata.

The Dynamic Cache Monitor is an installable Web application that displays simple cache statistics, cache entries and cache policy information. For detailed information, see Chapter 10, "Dynamic caching" on page 501.

14.9 Monitoring the IBM HTTP Server

The IBM HTTP Server server-status page is available on all supported IBM HTTP Server platforms. It shows performance data on a Web page in HTML format.

Perform the following steps to activate the server-status page:

1. Open the IBM HTTP Server file httpd.conf in an editor.

2. Remove the comment character “#” from the following lines:

```
#LoadModule status_module modules/mod_status.so

#<Location /server-status>
#   SetHandler server-status
#   Order deny,allow
#   Deny from all
#   Allow from .example.com
#</Location>
```

3. Customize the “.example.com” in the sample configuration to match your source workstation reverse DNS entry so you’re allowed to access the page. In our example this must be changed to:

```
Allow from .itso.ibm.com
```

4. Save the changes and restart the IBM HTTP Server.
5. Open the URL `http://<yourhost>/server-status` in a Web browser, and click **Refresh** to update the status.

If your browser supports refresh, you can also use the URL `http://<yourhost>/server-status?refresh=5` to refresh every 5 seconds. As shown in Figure 14-20 on page 828, you can see (along with additional information) the number of requests currently being processed, and the number of idle servers.

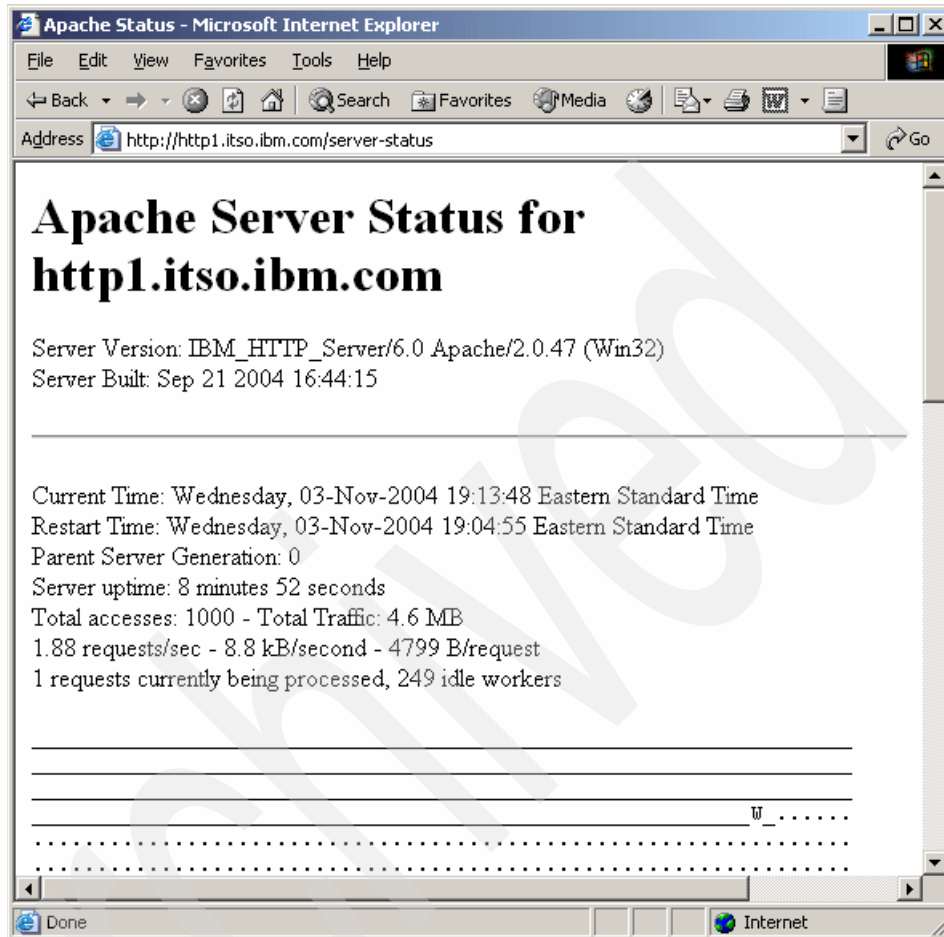


Figure 14-20 IBM HTTP Server status page

14.10 Log Analyzer

The Log Analyzer is a GUI tool that allows the user to view any logs generated with the loganalyzer TraceFormat, such as the IBM service log file and other traces using this format. It can take one or more service logs or trace logs, merge all the data, and display the entries in sequence.

Log Analyzer is shipped with an XML database, the *symptom database*, which contains entries for common problems, reasons for the errors, and recovery steps. The Log Analyzer compares every error record in the log file to the internal set of known problems in the symptom database and displays all the matches.

This allows the user to get error message explanations and information such as why the error occurred and how to recover from it. Figure 14-21 shows the conceptual model of Log Analyzer.

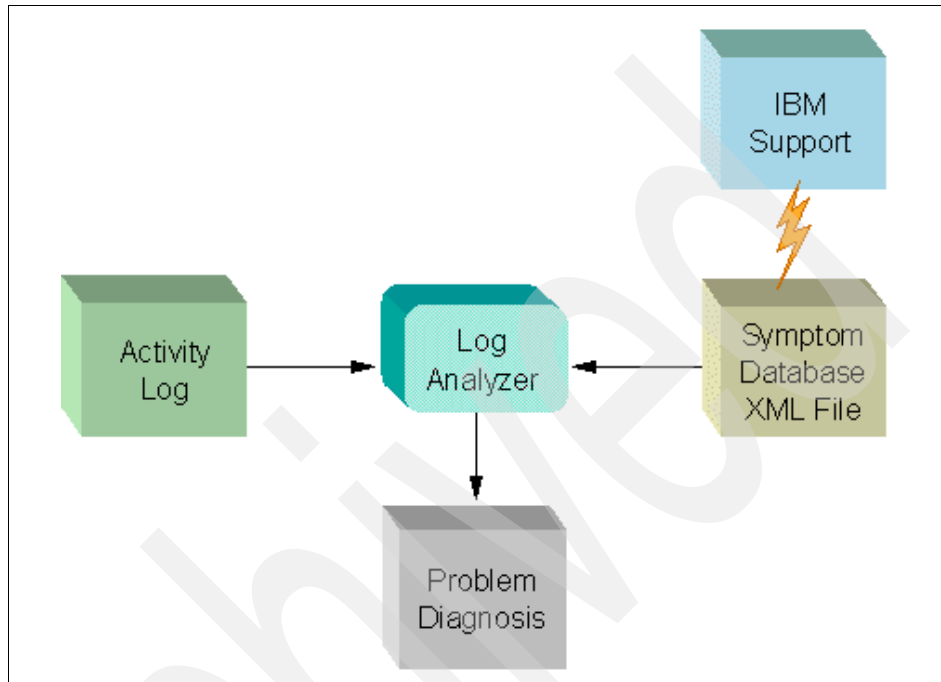


Figure 14-21 Conceptual model of Log Analyzer

For a detailed description about how to use Log Analyzer, refer to Section 9.5 of the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

14.11 Application management

The tooling supplied with the IBM WebSphere Application Server products provides facilities to monitor the applications you deploy on it, but they provide no cross-platform or cross-enterprise view of all your deployed applications. This is where application management tools come into play. They provide a transactional view over multiple servers and platforms throughout the enterprise.



Figure 14-22 Application management composition

Application management tools contain functionality to do problem determination and performance management for your applications, as shown in Figure 14-22.

These products are not supposed to replace any of the following tools:

- ▶ Source code profiling
- ▶ Systems management framework
- ▶ System resource management

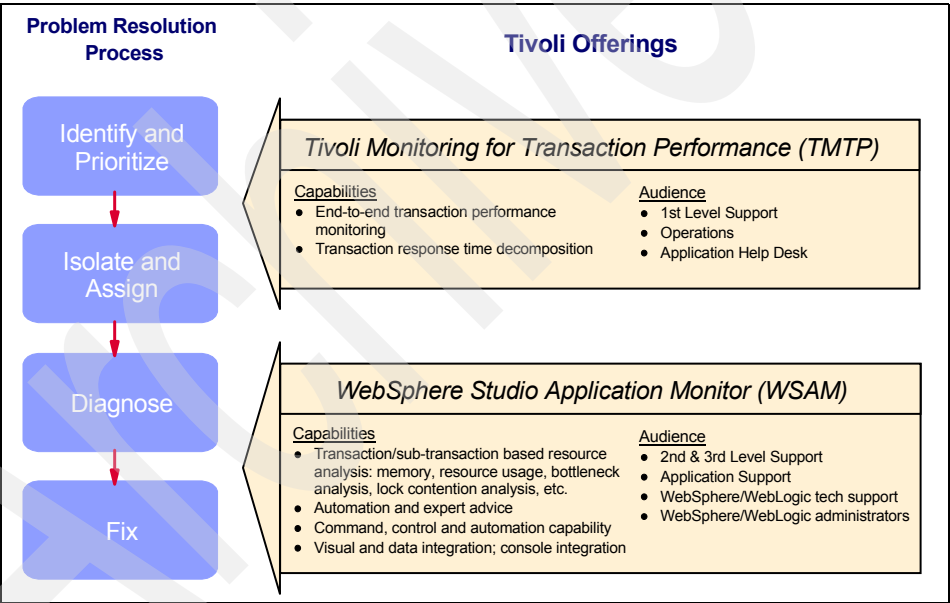


Figure 14-23 Tivoli application management offerings

As depicted in Figure 14-23 the problem and resolution process can be split into the following steps:

1. Identify and prioritize
2. Isolate and assign
3. Diagnose
4. Fix

These steps are supported by two Tivoli products that deliver application management functionality to a specific group of users:

- ▶ Tivoli Monitoring for Transaction Performance (TMTP)
Used by 1st level support, operations, and application helpdesks to isolate and identify performance problems through many technologies and across multiple servers.
- ▶ WebSphere Studio Application Monitor (WSAM)
Used by SMEs (Subject Matter Experts) to drill down into J2EE applications to find the source of an application performance problem.

The application management products WebSphere Studio Application Monitor and TMTP have the following unique capabilities:

- ▶ No manual instrumentation or application reengineering required
- ▶ Real-time problem determination with extensive drilldown capabilities
- ▶ Instance-level data capture correlates specific transactions with an end user experience
- ▶ Pinpoint problems that are masked with only aggregated data

14.11.1 Tivoli Monitoring for Transaction Performance V5.3 (TMTP)

Important: This section is based on Tivoli Monitoring for Transaction Performance V5.3 supporting WebSphere Application Server V5.1. At the time of writing this redbook, WebSphere V6 was not yet supported.

To start out with application management, you need a tool that provides you with the most insights about your environment. Today, these environments usually do not reside on a single machine or span only a single application server. Therefore, Tivoli Monitoring for Transaction Performance is able to track transactions over multiple machines and multiple application servers.

In addition, it is also capable of finding potential network problems and include network latency in transaction decomposition.

For prerequisite information, please refer to the InfoCenter at:

<http://publib.boulder.ibm.com/infocenter/tiv3help/topic/com.ibm.itmtp.doc/t pimst14.htm>

Tivoli Monitoring for Transaction Performance can monitor applications in two ways:

- ▶ Passive monitoring
- ▶ Active monitoring

Passive monitoring

This monitoring mode measures real client transactions and uses them for analysis. Tivoli Monitoring for Transaction Performance has the following passive monitoring capabilities:

- ▶ ARM instrumentation of servers (Web, application, in-house or third-party)
In this mode TMTP shows the transactions within ARM enabled applications and tracks them through the various components and composite parts.
- ▶ High-level decomposition using a reverse proxy
In this mode TMTP is able to measure end-to-end response times including user-display times in the browser by acting as a reverse proxy. This measures the real user-experience and not only the response time of the server components.

Active monitoring

During active monitoring Tivoli Monitoring for Transaction Performance replays recorded user transactions on the infrastructure. During this replay the response times throughout the infrastructure are measured and compared to thresholds.

In addition to testing the infrastructure with pre-defined use-cases this can also be used to test the same use-cases from multiple locations throughout the corporate network or throughout the Internet.

Tivoli Monitoring for Transaction Performance has the capability to replay transactions by using any of the following methods:

- ▶ URL-record and playback using Synthetic Transaction Investigator
- ▶ Windows client-record and playback using Rational Robot GUI scripts
- ▶ Protocol record and playback using Rational Robot VU scripts

Request decomposition

During request decomposition support staff can find problems in parts of the infrastructure or application components. Tivoli Monitoring for Transaction Performance provides a graphical view of the transaction as it is processed throughout the infrastructure and provides timing information for all sub-components of the transaction. TMTPs discovery capabilities allow you to exactly see which application modules (EJBs, Servlets, JSPs, etc.) are called.

V5.3 also collects detailed transaction information from DB2 V8.2 when accessed via JDBC.

You can see an example from the pet store sample shop in Figure 14-24. It shows a cart request coming into the J2EE infrastructure to a servlet. The servlet then displays the output by forwarding the request to a JSP. Both of these components use JDBC to connect to the back-end datastore.

The red triangle above the request URI object shows that this transaction was above the threshold set by the administrator.

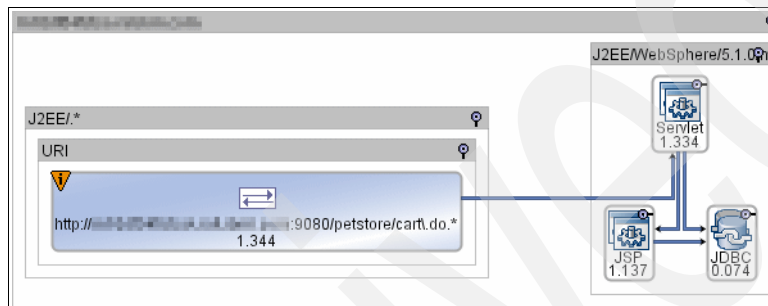


Figure 14-24 TMAP pet store sample request decomposition

This simple example already shows the power of Tivoli Monitoring for Transaction Performance applied to a sample application.

Tip: The latest version of Tivoli Monitoring for Transaction Performance now also supports Web services transaction decomposition.

Integration points

Tivoli Monitoring for Transaction Performance integrates with the following IBM products:

- ▶ Tivoli Intelligent Orchestrator
- ▶ IBM Business Workload Manager
- ▶ Event Integration (TEC, SNMP, e-mail, command scripts)
- ▶ Tivoli Web Health Console
- ▶ Tivoli Data Warehouse (18 Crystal reports included with TMAP)

14.11.2 WebSphere Studio Application Monitor V3.1 (WSAM)

The capabilities provided by the Tivoli Monitoring products and the WebSphere tooling still lack functionality vital for subject matter experts in L2 or L3 operations support groups.

One of the products that was designed to address the needs of these operations groups was developed by Cyanea, which was acquired by IBM, and becomes part of the Tivoli brand.

WebSphere Studio Application Monitor monitors applications running on WebSphere and WebLogic and presents a correlated view of composite, mixed workload transactions from J2EE to CICS, IMS, and WebSphere MQ back ends.

For prerequisite information, please refer to the IBM Web site at

<http://www.ibm.com/software/awdtools/studioapplicationmonitor/sysreq/>

The key focus areas include:

- ▶ Application hung/slow conditions
- ▶ Intermittent application faults
- ▶ Application memory leaks
- ▶ Alert generation for specific application events
- ▶ Deployment resource estimation from an application perspective
- ▶ “Composite” application transaction profiling (for example, J2EE-J2EE or J2EE-mainframe)

Architecture and design

WebSphere Studio Application Monitor is made up of multiple components connected across TCP/IP networks. It has a loosely coupled structure with discrete components, many of which can be started up and brought down independently of one another. This means that the product is very scalable, potentially supporting thousands of networked servers through single or multiple operational points of control. Figure 14-25 on page 835 illustrates WebSphere Studio Application Monitor's topology, identifying the major components and where they reside.

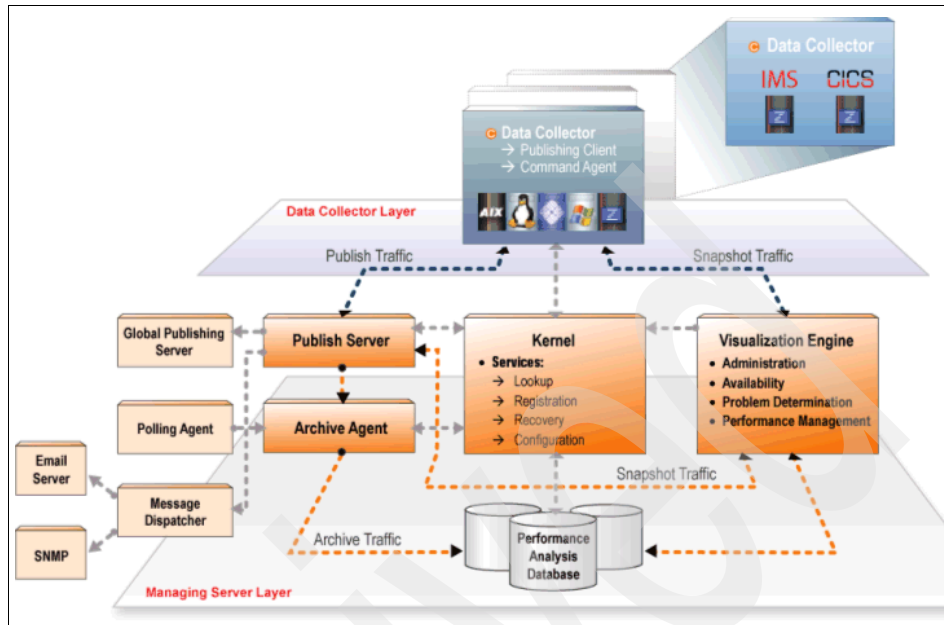


Figure 14-25 WebSphere Studio Application Monitor architecture

There are three principal parts to WebSphere Studio Application Monitor: a managing server which must reside on a distributed platform or on a zLinux system, distributed data collectors, and z/OS® data collectors. The managing server houses a relational database like DB2 UDB, an instance of WebSphere Application Server running the WebSphere Studio Application Monitor console application, and Java-based overseer components that control the managing server itself and the storage of monitoring data. As might be expected, very I/O-intensive operations go on in the managing server. Moving these processes to the managing server means that they do impact the performance of the monitored servers or their platforms.

The data collectors are built for dexterity and speed. They unleash probes that gather up monitoring data about applications running in J2EE servers or in legacy systems like CICS or IMS. Each data collector is written to take advantage of speedy and efficient services available on the host operating system.

Note: As mentioned, the managing server requires an underlying WebSphere Application Server as it runs as an application on that application server instance. Currently, the managing server requires an underlying WebSphere V5.1 server but the data collectors are able to monitor WebSphere V6 servers. The managing server software includes a Websphere Application Server V5.1 license.

Once at the managing server, the monitoring data is prepared for real-time displays within the monitoring console and is inserted into the WebSphere Studio Application Monitor data repository. These are very resource-intensive operations; moving them to a stand-alone distributed server (or servers) isolates them from other enterprise activities, thus reducing WebSphere Studio Application Monitor's footprints in the monitored systems. This design also helps keep WebSphere Studio Application Monitor's processing overhead at levels low enough for 24x7 production system monitoring.

Key features

WebSphere Studio Application Monitor provides the following key features:

- ▶ **System Resources**
 - Displays summary information for the system resources on the selected application server
 - JVM CPU Usage, JVM Memory Usage, EJB Coverage, EJB Activity, Transactions Initiated and JSP/Servlet Activity and Coverage among the hundreds of captured metrics
- ▶ **Trap and Alert Management**

A trap can be set with a threshold on a specific system or application level event or behavior. When the system meets the criteria of the trap, the action (alert) occurs.
- ▶ **Software Consistency Check**

Detects software mismatches in “cloned” runtime environments and supports environmental and application level detection.
- ▶ **Account and Server Group Management**
 - A “server group” consists of a user-defined collection of servers.
 - Accounts may be associated with specific groups.
 - Access to data and operations of the group can be restricted.
- ▶ **Monitoring On Demand**
 - Monitoring scope and granularity of information returned may be changed without restarting either the applications or the application servers.
 - No need to pinpoint specific classes or methods in advance; there is no need to designate what needs to be monitored.
 - Three discreet monitoring levels available.

For more details please refer to “Monitoring on Demand” on page 837.
- ▶ **Memory Analysis**

- In-depth analysis of metrics including JVM memory utilization, Garbage Collection (GC) frequency, time spent in GC and heap contents.
- Memory leak detection and analysis broken down by Java class.
- ▶ WebSphere Studio Application Monitor Portal Monitoring
 - Deep insight into portal environments throughout development and deployment life cycle.
 - Manage and monitor key portal performance indicators in production.
 - Full correlation of J2EE portlets and legacy transactional environments with CICS and IMS.

Monitoring on Demand

This unique feature of WebSphere Studio Application Monitor does not only allow you to perform changes of monitoring scope and granularity during production without restarting either the applications or the application servers, but also makes it possible to automatically change these levels without human intervention.

Features/Information available at different monitoring levels:

- ▶ Common to all Levels
 - Availability management
 - System resources
- ▶ L1: Request
 - CPU information
 - Elapsed time
- ▶ L2: Component (JDBC/SQL, EJB, CICS, RMI)
 - L1 information for component
- ▶ L3: Java Method (Entry/Exit)
 - L2 information for Java methods

Note: Certain traps and alerts need L3 to capture method trace.

Integration points

WebSphere Studio Application Monitor integrates into an existing Tivoli or BMC Patrol monitoring infrastructure by sending SNMP traps of events monitored to the responsible operations groups.

14.12 Reference

For more information about the topics covered in this chapter, refer to:

- ▶ e-Pro mag.com - Ruth Willenborg: Monitoring Performance with WebSphere:
<http://www.e-promag.com/eparchive//index.cfm?fuseaction=viewarticle&ContentID=1492&publicationid=13&PageView=Search&channel=2>
- ▶ IBM WebSphere Developer Technical Journal: Writing PMI applications using the JMX interface:
http://www.ibm.com/developerworks/websphere/techjournal/0402_qiao/0402_qiao.html
- ▶ WebSphere Performance Diagnostics - Going beyond the Metrics:
<http://www.sys-con.com/websphere/article.cfm?id=207>
- ▶ WebSphere Studio Application Monitor (WSAM) Redbooks:
 - *Planning for the Installation and Rollout of WebSphere Studio Application Monitor 3.1*, SG24-7072
 - *Installing WebSphere Studio Application Monitor V3.1*, SG24-6491
- ▶ Tivoli Monitoring for Transaction Performance V5.3 (TMTP) redbook:
End-to-End e-business Transaction Management Made Easy, SG24-6080



Development-side performance and analysis tools

This chapter discusses tools for development-side measuring and performance analysis WebSphere applications.

In particular, we discuss the following:

- ▶ The profiling tools incorporated into IBM Rational Application Developer V6.0, called the Profiler throughout this chapter.
- ▶ IBM Page Detailer, which is available as a separate product.

15.1 Introduction

Performance of an application must be a focus area throughout the project cycle. Traditionally, performance testing has occurred late in the cycle, towards the end of the testing phase, or as part of the deployment process. This does not leave much time to identify and fix any performance issues, particularly if significant architecture changes are required. It is much more difficult and time consuming to improve performance in the later phases of the project cycle than to ensure the application performs adequately from the beginning. To support this, IBM Rational Application Developer V6 includes facilities for profiling and measuring performance of Web applications. These tools can be used during the coding phase of the development cycle to identify problems earlier.

Two of the tools that are available to measure and analyze performance are the *profiling tools (Profiler)* which are part of IBM Rational Application Developer V6, and *Page Detailer*, which is downloadable from IBM alphaWorks at:

<http://www.alphaworks.ibm.com/tech/pagedetailer>

The profiling tools help with the analysis of the runtime behavior of aspects of the application, such as execution time and flows and memory usage. The Page Detailer provides information about the client experience when accessing the application, such as response times. See 15.2, “The Profiler (profiling tools)” on page 840 and 15.3, “IBM Page Detailer” on page 886 for details on these products.

The application we have been using to demonstrate the Profiler is the Trade 6 sample application. See 8.8, “Installing and configuring Trade 6” on page 436 for more information about Trade 6.

This chapter gives you basic information about the Profiler. If you need detailed information about the IBM Rational Application Developer V6, then you should read the *Rational Application Developer V6 Programming Guide*, SG24-6449.

15.2 The Profiler (profiling tools)

The objective of profiling is to assist developers recognize and isolate a variety of performance problems before the deployment of their application into a production environment. Traditionally, performance profiling is done once an application is getting close to deployment or when it has already been deployed. Using the profiling tools in IBM Rational Application Developer V6 allows the developer to move this analysis to a much earlier phase in the development cycle, therefore giving you more time to modify your application or architecture based on any problems detected.

The types of problems that the IBM Rational Application Developer V6 profiling tools can assist in detecting include:

- ▶ Memory leaks
- ▶ Performance bottlenecks
- ▶ Excessive object creation
- ▶ System resource limits

The profiling tools can be used to gather performance information about applications that are running:

- ▶ Inside an application server, such as WebSphere Application Server
- ▶ As a stand-alone Java application
- ▶ On the same machine as IBM Rational Application Developer V6
- ▶ On a remote machine from IBM Rational Application Developer V6
- ▶ In multiple JVMs

15.2.1 What's new with profiling

IBM Rational Application Developer V6 has introduced additional features into profiling to assist the developer to perform improved profiling analysis of their code. These can be classified in a number of distinct areas:

- ▶ Memory analysis
- ▶ Thread analysis
- ▶ Execution time analysis
- ▶ Code coverage
- ▶ Probekit

Memory analysis

The memory analysis capability in IBM Rational Application Developer V6 has been enhanced with the addition of new views described in Table 15-1, and new capabilities found in Table 15-2 on page 842.

Table 15-1 New Memory Analysis views

View Name	Description
Leak Candidates view	A tabular view to assist the developer in identifying the most likely objects responsible for leaking memory.
Object Reference Graph view	A graphical view that shows the referential relationship of objects in a graph highlighting the allocation path of leak candidates.

Table 15-2 New Memory Analysis capabilities

Capability	Description
Memory Leak Analysis - Manual	Allows at the discretion of the developer to capture memory heap dumps after application warm-up. That is, when classes are loaded and initialized.
Memory Leak Analysis - Automatic	Provides timed memory heap dumps at specified intervals while the Java application is running.

Thread analysis

The thread analysis capability in IBM Rational Application Developer V6 offers the view listed in Table 15-3.

Table 15-3 New Thread Analysis views

View Name	Description
Thread view	A graphical view of all threads available, their state and which thread is holding locks. It assists in identifying thread contentions.

Execution time analysis

The execution time analysis has been enhanced with the addition of views described in Table 15-4.

Table 15-4 New Execution time analysis views

View Name	Description
Performance Call Graph view	A graphical view focusing on data that indicates potential performance problems including statistical information.
Method Details view	A view that provides complete performance data for the currently displayed method, including information about its callers and descendants.

Code coverage

Code coverage is a new capability in IBM Rational Application Developer V6.0. It is used to detect areas of code that have not been executed in a particular scenario that is tested. This capability is a useful analysis tool to integrate with

component test scenarios and can be used to assist in identifying test cases that may be missing from a particular test suite or code that is redundant.

New views associated with this capability are shown in Table 15-5.

Table 15-5 New views associated with code coverage

View Name	Description
Coverage Navigator	A graphical view that shows coverage levels of packages, classes and methods and their coverage statistics.
Annotated Source	Includes displays which: <ul style="list-style-type: none">▶ Have a copy of the code marked indicated tested, untested and partially tested lines.▶ Shows at the class and method level a pie chart with the line coverage statistic.
Coverage Statistics	A tabular view showing the coverage statistics.

Probekit

The probekit is a new capability that has been introduced into IBM Rational Application Developer V6.0. It is a scriptable byte-code instrumentation (BCI) framework, to assist in profiling runtime problems by inserting Java code fragments into an application. The framework is used to collect detailed runtime information in a customized way.

A probekit file can contain one or more probes with each containing one or more probe fragments. These probes can be specified when to be executed or on which program they will be used. The probe fragments are a set of Java methods that are merged with standard boilerplate code with a new Java class generated and compiled. The functions generated from the probe fragments appear as static methods of the generated probe class.

The probekit engine called the BCI engine is used to apply probe fragments by inserting the calls into the target programs. The insertion process of the call statements into the target methods is referred as *instrumentation*. The data items requested by a probe fragment are passed as arguments (for example the method name and arguments). The benefit of this approach is that the probe can be inserted into a large number of methods with small overhead.

Probe fragments can be executed at the following points (see IBM Rational Application Developer V6's online help for a complete list):

- ▶ On method entry or exit
- ▶ At exception handler time
- ▶ Before every executable code when source code is available
- ▶ When specific methods are called, not inside the called method

Each of the probe fragments can access the following data:

- ▶ Package, class, and method name
- ▶ Method signature
- ▶ This object
- ▶ Arguments
- ▶ Return value
- ▶ The exception object that caused an exception handler exit to execute, or an exception exit from the method

There are two major types of probes available to the user to create, as described in Table 15-6.

Table 15-6 Types of probes available with Probekit

Type of Probe	Description
Method Probe	Probe can be inserted anywhere within the body of a method with the class or jar files containing the target methods instrumented by the BCI engine.
Callsite Probe	Probe is inserted into the body of the method that calls the target method. The class or jar files that call the target instrumented by the BCI engine

15.2.2 Profiling architecture

The profiling architecture that exists in IBM Rational Application Developer V6 has originated from the data collection engine feature provided by the open source Eclipse Hyades project found at:

<http://www.eclipse.org/hyades>

Hyades provides the IBM Rational Agent Controller daemon a process for enabling client applications to launch host processes and interact with agents that exist within host processes. Figure 15-1 on page 845 depicts the profiling architecture.

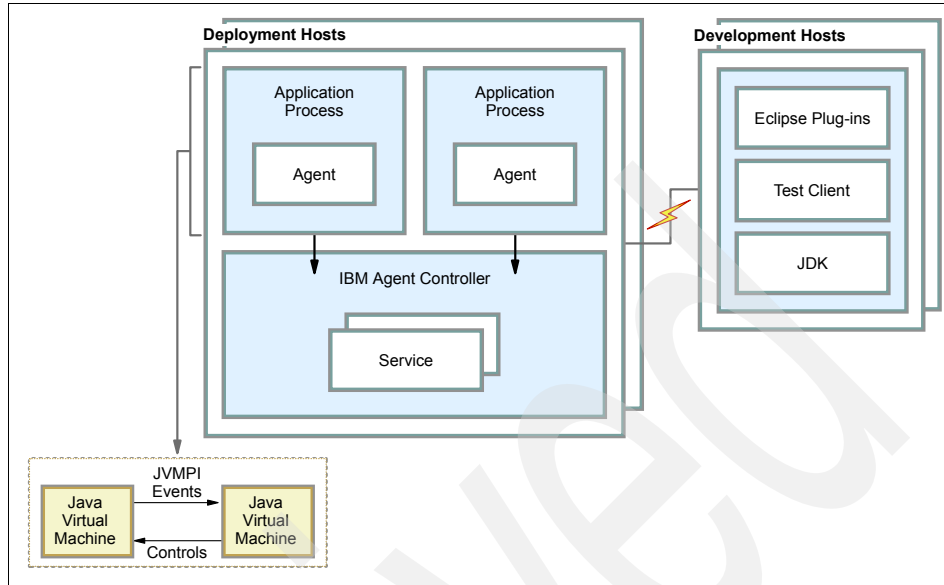


Figure 15-1 Profiling architecture of IBM Rational Application Developer V6

The definitions of the profiling architecture are as follows:

► Application process

The process that is executing the application consisting of the Java Virtual Machine (JVM) and the profiling agent.

► Agent

The profiling component installed with the application that provides services to the host process, and more importantly, provides a portal by which application data can be forwarded to attached clients.

► Test client

A local or remote application that is the destination of host process data that is externalized by an agent. A single client can be attached to many agents at once but does not always have to be attached to an agent.

► IBM Rational Agent Controller

A daemon process that resides on each deployment host providing the mechanism by which client applications can either launch new host processes, or attach to agents coexisting within existing host processes. The Agent Controller can only interact with host processes on the same node.

- Deployment hosts

The host that an application has been deployed to and is being monitored by a profiling agent.

- Development hosts

The host that runs an Eclipse compatible architecture such as IBM Rational Application Developer V6 to receive profiling information and data for analysis.

Each application process shown in Figure 15-1 on page 845, represents a JVM that is executing a Java application that is being profiled. A profiler agent is attached to each application to collect the appropriate runtime data for a particular type of profiling analysis. This profiling agent is based on the Java Virtual Machine Profiler Interface (JVMPi) architecture. More details on the JVMPi specification can be found at:

<http://java.sun.com/j2se/1.4.2/docs/guide/jvmpi>

The data collected by the agent is then sent to the agent controller, which then forwards this information to IBM Rational Application Developer V6 for analysis and visualization.

There are two types of profiling agents available in IBM Rational Application Developer V6:

- Java Profiling Agent
- J2EE Request Profiling Agent

See IBM Rational Agent Controller below for more information about these agents.

15.2.3 IBM Rational Agent Controller

As mentioned before, the profiling tools work in conjunction with the IBM Rational Agent Controller, which is a separately installable program from the IBM Rational Application Developer V6 installation. Thus you need to make sure that the Agent Controller has been installed on the host where you want to monitor the process.

The Agent Controller is a daemon process that enables client applications to launch host processes and interact with agents that coexist within host processes. The Agent Controller interacts with the following components:

- Host process

The host process contains the application that is to be profiled. This can be a stand-alone Java process or a WebSphere Application Server instance

(including an instance running in an IBM Rational Application Developer V6 test environment). An Agent Controller must be executing on the same machine as the host process.

► Agent

An agent is a reusable binary file that provides services to the host process, and more importantly, provides a portal by which application data can be forwarded to attached clients. A host process can have one or more agents currently running within it.

– Java Profiling Agent

The Java Profiling Agent uses the Java Virtual Machine Profiler Interface (JVMPI). The agent is a library that provides services to the host process to capture and record the behavior of a Java application, and makes this data available to attached clients. It is used for the collection of both stand-alone Java applications as well as applications running on an application server. Figure 15-1 on page 845 shows the use of the Java Profiling Agent.

– J2EE Request Profiler

The J2EE Request Profiler is an agent that operates in a WebSphere Application Server process. It collects runtime data relating to the execution of the Web application, such as execution of EJBs and servlets, by intercepting requests to the EJB or Web containers. Distributed environments can be profiled by connecting to J2EE Request Profiler agents running on different nodes.

The J2EE Request Profiler only provides a subset of the information available from the Java Profiling Agent, since it only looks at the execution of the enterprise components, and not at Java objects and methods that are called from the EJBs and servlets. The J2EE Request Profiler does, however, have the ability to combine data from more than one WebSphere instance, allowing you to analyze and measure the behavior and performance of a distributed application.

► Client

A client is a local or remote application that retrieves and displays data that is collected by an agent. A single client can be attached to many agents at once, and can connect to Agent Controllers on local or remote machines. All communication between the client and agents occurs via the Agent Controller. A profiler client is provided with IBM Rational Application Developer V6. The profiling functionality in Rational Application Developer can be accessed from the *Profiling and Logging Perspective*.

In this chapter, we discuss the profiling of an application running on WebSphere Application Server. Similar techniques can be used to profile applications running in a Rational Application Developer test environment.

Before attempting to profile an application, ensure that the Agent Controller has been started. In Windows this can be checked from the Services program.

15.2.4 Setting up for Profiling

The prerequisites to be performed for profiling are as follows:

- ▶ Install IBM Rational Application Developer V6 on the development host.
- ▶ Install IBM Rational Agent Controller on the deployment host or on the development host if they are one and the same.

Enable Profiling in WebSphere Application Server

The default configuration of WebSphere Application Server V6 has the J2EE Request Profiling agent generation of sequence diagrams disabled. Users requiring this feature need to enable it by performing the following:

For a remote Application Server Instance:

To configure your remote application server for profiling, do the following:

1. In the Administrative Console select **Servers -> Application servers -> <AppServer_Name>**.
2. Under the Server Infrastructure category choose **Java and Process Management-> Process Definition**.
3. Select **Java Virtual Machine** from the Additional Properties pane. Enter **-XrunpiAgent -DPD_DT_ENABLED=true** into the Generic JVM arguments field and click **OK**.

Note:

- ▶ The **-XrunpiAgent** parameter enables the Java Profiling Agent.
- ▶ The **-DPD_DT_ENABLED=true** parameter is for the J2EE Request Profiler.

4. Save the configuration and restart the server(s).

For an application server in the test environment:

To profile an application in IBM Rational Application Developer V6, the server in the test environment must be started in *profiling mode*. From the Server or J2EE Perspective, right-clicking the server displays a menu that includes the profiling option. This is shown in Figure 15-2 on page 849. Another prerequisite is that you have deployed the application to the server and configured any server

resources required. We suggest that you perform some basic testing of the application in Rational Application Developer, prior to using the profiling tools, to ensure that the application is operating correctly.

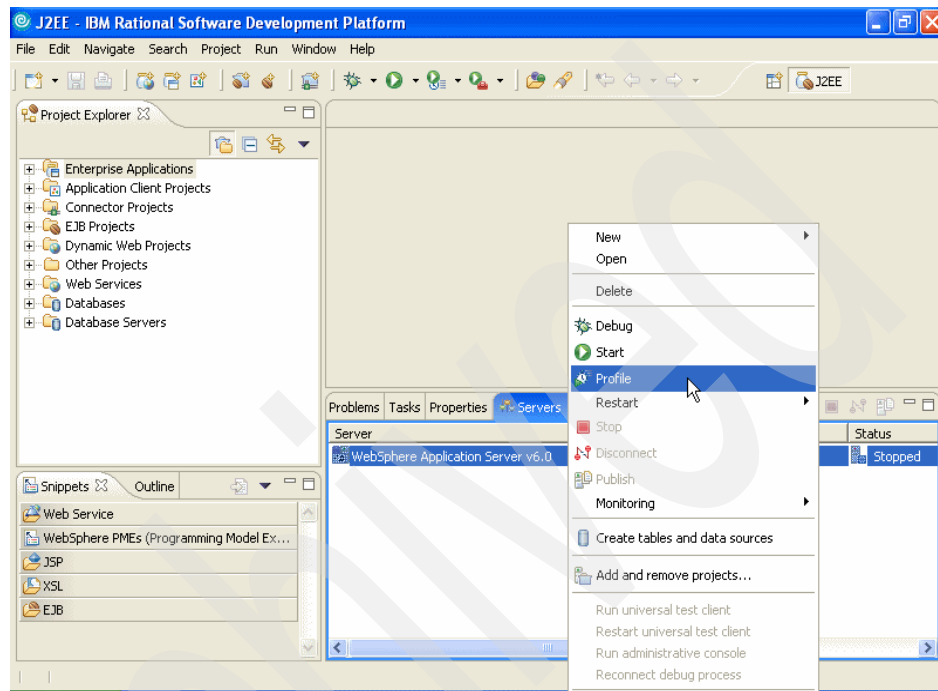


Figure 15-2 Starting a server in profiling mode

Once the server has been started, you have to configure profiling in IBM Rational Application Developer V6.

Configure profiling in Rational Application Developer

The remainder of this procedure is the same for both local and remote profiling. A window is displayed to select the Java process to attach to. The process ID is the one for the WebSphere Application Server instance and it can be found in the .pid file created in the server root, when the server is started.

1. In IBM Rational Application Developer V6, switch to the **Profiling and Logging** perspective:
 - a. Select **Window -> Open Perspective -> Other**.
 - b. In the Select Perspective window, first select the **Show all** checkbox to make the Profiling and Logging view available for selection.

- c. Select **Profiling and Logging** and click **OK**. Answer **OK** to the following question regarding enablement of Profiling and Logging.
2. Attach to a Remote Java process for the server instance:
 - a. Select **Run -> Profile....** In the following configuration window double-click **Attach - Java Process**.
 - b. If desired, enter a name for your configuration. See Figure 15-3 on page 851.
 - c. Enter the name of the host you want to perform profiling on into the Host name or IP address field. If the Agent Profiler was installed with default options, the port number should be correct. Click **Add**.
 - d. Click **Test Connection** to verify that the communication works. If the connection fails, check if the Agent Controller is running on the target server and if it is using the default port.

Tip: If you are using this functionality for the first time, maybe you won't get the choice "Attach - Java Process" in the Configurations pane. If this is the case, do the following:

Select **Window -> Preferences**. Expand **Profiling and Logging**. Click **Hosts**. Define the host you want to profile. Test the connection, then click **OK**. Next select **Run -> Profile** once again and this time you should see the Attach - Java Process selection.

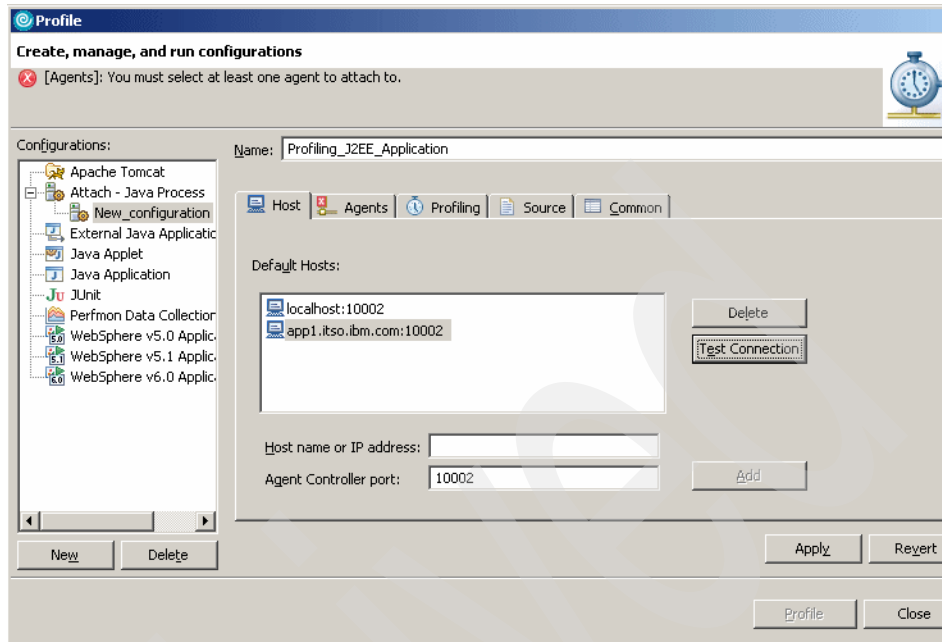


Figure 15-3 Configuring the profiling host

3. Next select the **Agents** tab. There are two agents available: Java Profiling Agent and J2EE Request Profiler.

Troubleshooting:

- ▶ If the Java Profiling Agent is missing, the `-XrunpiAgent` parameter is missing in the generic JVM arguments.
- ▶ If the J2EE Request Profiler is not available check if `-DPD_DT_ENABLED=true` has been set.

See “Enable Profiling in WebSphere Application Server” on page 848.

To obtain detailed information about the runtime behavior of the application, select the **Java Profiling Agent** option as shown in Figure 15-4 on page 852. The J2EE Request Profiler will provide higher-level overview data about the application. It is also possible to select both agents from this window.

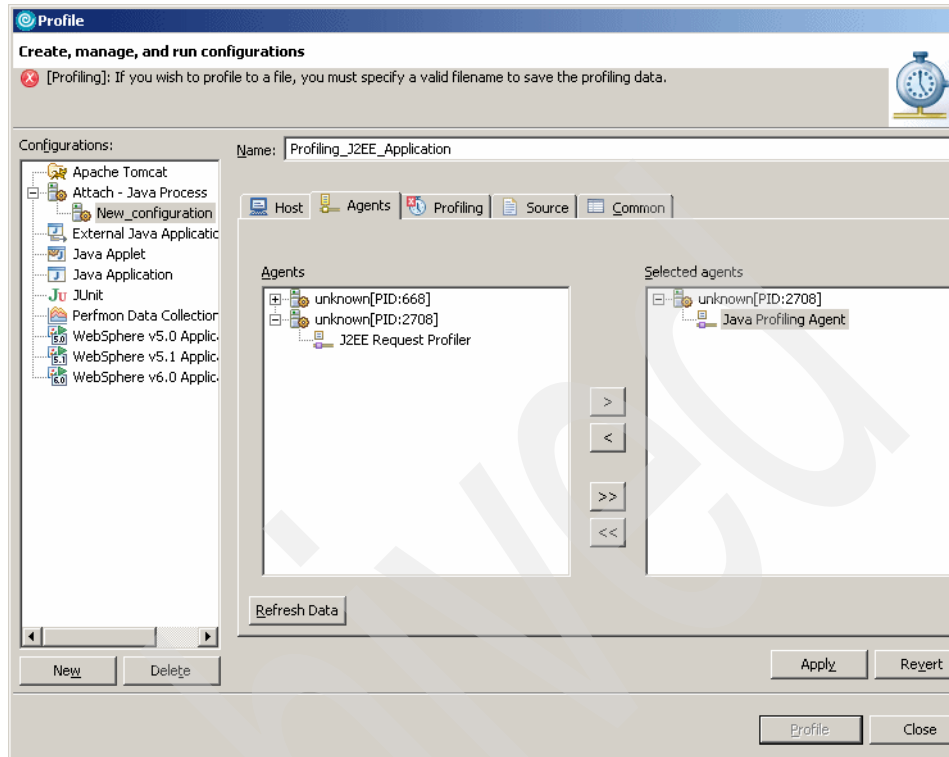


Figure 15-4 Select the agents associated with the Java processes

Important: When studying the execution flow of an application, we recommend that the J2EE Request Profiler be used.

On the other hand, to obtain detailed information related to instance sizes, we suggest using the Java Profiling Agent with Select instance level information checked (you need to Edit a profiling set to find this checkbox).

4. Click the **Profiling** tab to obtain the window for specifying the profiling project, the monitoring limits and the monitor for saving the profiler data.
 - a. On the Overview tab, the profiling sets can be defined. There are a number of predefined filter sets available for analyzing execution time, code coverage, finding memory leaks and so on. It is also possible to define your own profiling sets.

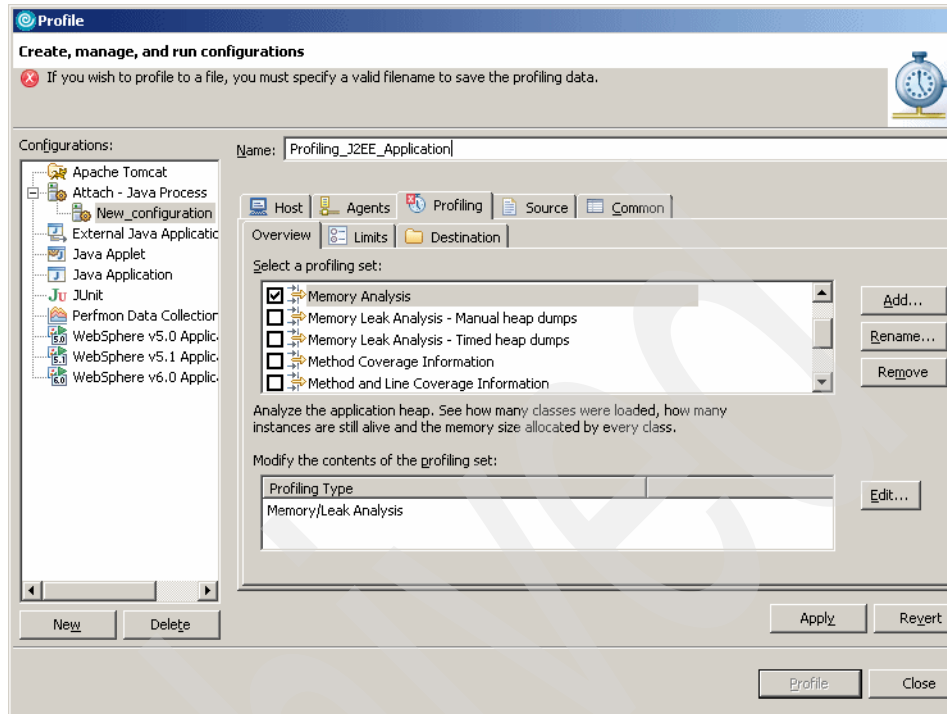


Figure 15-5 Profiling sets

Tips:

- If you want to profile Trade 6, it is important to define some additional filter settings. This is because Trade 6 belongs to `com.ibm.*` which is excluded from profiling by default. Therefore, on the Overview tab, click **Edit...** Click **Next** to display the filters. In the Contents of selected filter set pane click **Add...**

Enter `com.ibm.websphere.samples.trade.*` into the Package or Class field, `*` into the Method Name field and select **INCLUDE** from the Rule drop-down box. Click **Finish**.

- The filters also need to be changed accordingly for profiling other WebSphere sample applications.

- Select the **Limits** tab to display the panel shown in Figure 15-6 on page 854 where you can limit the amount of data collected by the Profiler. It allows you to instruct the Profiler to stop profiling after a particular number of method invocations, or after a specified time has elapsed.

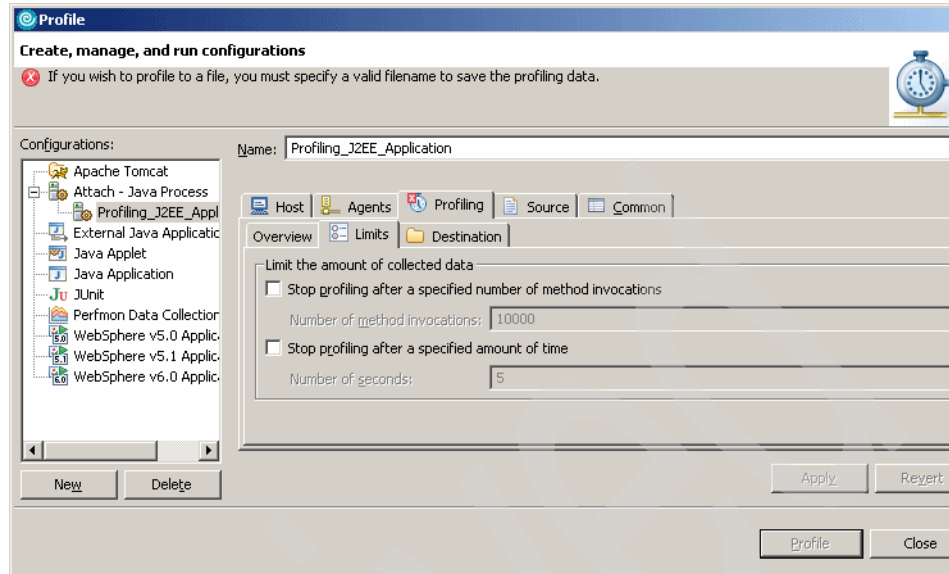


Figure 15-6 Limit the amount of data collected

- c. Select the **Destination** tab. Here you have the ability to redirect output to a file. This is especially useful in situations where the information generated from the application being monitored exceeds the amount of memory that IBM Rational Application Developer V6 has been allocated or that is available on the system.
5. Click the **Profile** button to complete the procedure for attaching to the (remote) Java process.

Start data collection

At this point, the Profiler is configured but not yet collecting data. To start collecting data, select the appropriate agent in the Profiling Monitor pane, click the right mouse button, and choose **Start Monitoring** from the menu as shown in Figure 15-7 on page 855.

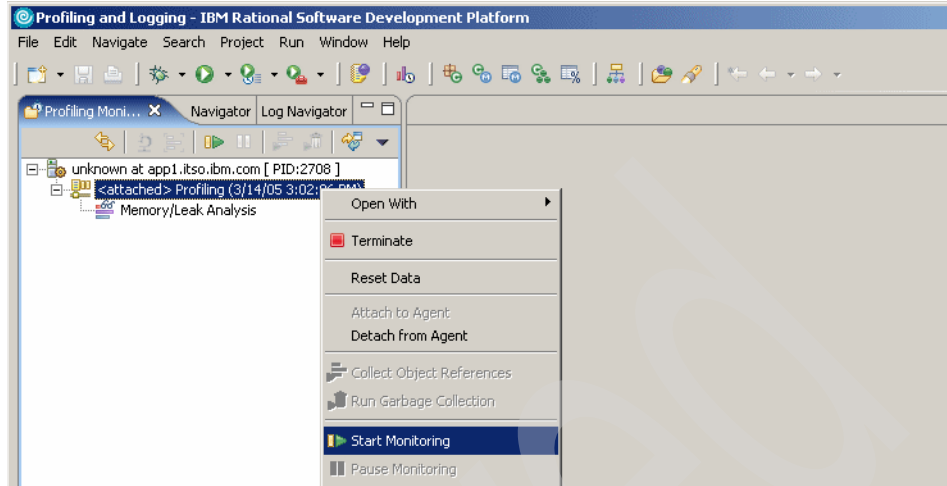


Figure 15-7 Start monitoring

Once monitoring has started, you are ready to start profiling the application.

Configuration settings for profiling

Note that the configuration settings for profiling can be accessed from the **Window -> Preferences** menu of the Workbench. They can be found on the **Profiling and Logging** page of the Preferences window as displayed in Figure 15-8 on page 856:

- ▶ Ensure that the **Enable profiling** check box is selected.
- ▶ If the **Show profiling tips** check box is selected, tips will be displayed periodically during your profiling session.
- ▶ The Default profiling project name can be changed if desired. This specifies under which project in IBM Rational Application Developer V6 the profiling data will be stored. Although the Agent Controller local port can be changed, this is usually not necessary. The port specified here must be the same as in the configuration for the Agent Controller. The default is 10002.
- ▶ Select **Profiling and Logging -> Profiling** for options to set default values for the profiling filters and options. Here you can limit the amount of data to be collected.
- ▶ Select **Profiling and Logging -> Appearance** to specify the format for the different views and graphs, such as Execution Flow Graph, Thread view, and UML2 Sequence Diagram. Here you can also specify the resources that will appear in the Profiling Monitor window. As a minimum, the **Processes**, **Profiling agents** and **Logging agents** should be selected (please note that

checking a box in this panel actually *hides* the selection from the Profiling Monitor view, therefore make sure that above selections are not checked).

- Select **Profiling and Logging -> Hosts** to specify a default list of hosts to be used when profiling a remote application.

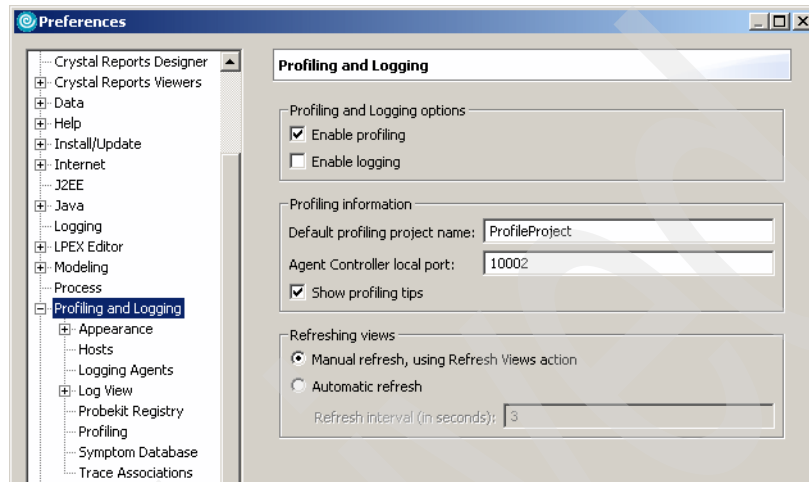


Figure 15-8 Preferences for profiling

15.2.5 Profiling an application

This section describes the tasks involved in profiling an application.

Testing methodology

There are a few different approaches that can be taken when profiling a Web application. One approach is to profile individual pieces of functionality (possibly as part of the unit test process as they are being developed). Alternatively, a mixture of operations that represent typical user behavior could be executed, either manually or using a load testing tool such as Rational Performance Tester V6.1 or OpenSTA. Initially the testing can cover a broad range of scenarios, and then as particular areas of functionality are identified, more detailed profiling can be done for them. The Trade 6 application includes a servlet “scenario” that simulates a set of users by executing a trade operation for a random user on each invocation of the servlet. A similar servlet or client program that exercises a broad range of application functionality may also be useful for other applications.

Using the Profiler

Once the Profiler has been configured, the application can be accessed via the normal URL and the profiling data will be captured. In order for the captured data to be displayed in the Profiling Perspective, right-click in the Profiling Monitor view and select **Refresh Views**. This will update the data for all views, not just the one that is currently displayed. Once the data has been displayed, the view has to be refreshed again in order to display newly captured data.

15.2.6 Profiler views

As outlined in 15.3.1, “Overview” on page 886, there are a number of different views that can be displayed in the Profiling Perspective. To open a new view, right-click in the Profiling Monitor pane and select one of the options from the **Open With** menu. Profiler views can also be displayed using the smarticons on the tool bar. The Profiler menu and toolbar are shown in Figure 15-9.

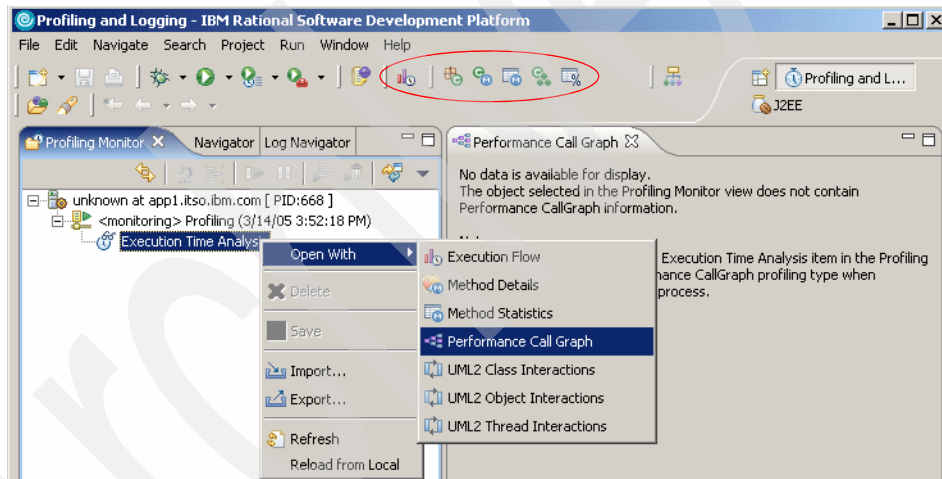


Figure 15-9 Profiler view menus

The choice of views available and their content depends on the context menu of the resource. You can make the resources visible either from **Window -> Preferences** as explained in “Configuration settings for profiling” on page 855 or from the perspective as shown in the Figure 15-10 on page 858.

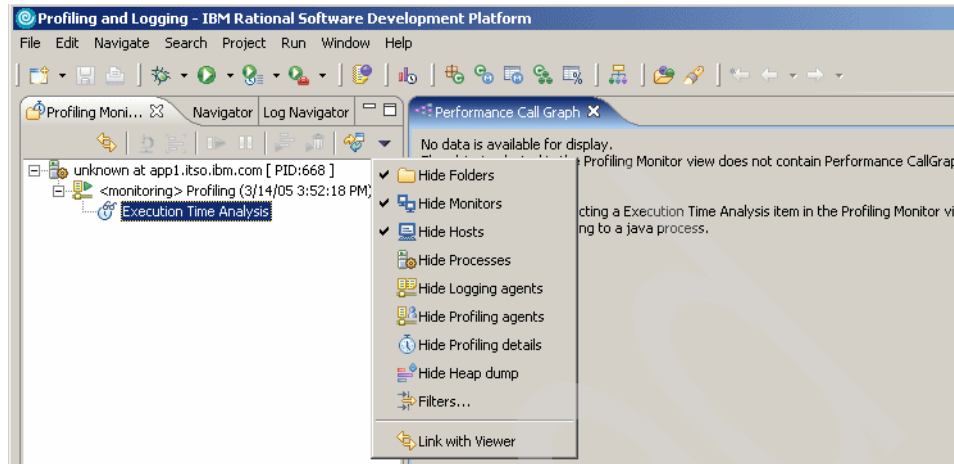


Figure 15-10 Selecting resources

Package Statistics table

The Package Statistics table displays the profiling data for the package with the ability to drill down to the class level (see Figure 15-11 on page 859).

This information allows you to get a high-level view of which packages are most frequently used, based on the frequency on which methods of classes belonging to the package are called and the number of instances of these classes. It also provides information about which packages use the most system resources in terms of execution time of methods for classes defined in the package and the memory used by object instances of the classes. It is possible to drill down to the class level for a particular package by clicking the + symbol displayed on the left of the package name. To reduce the amount of data being displayed, the output can be filtered based on the package name. These filters can include * as a wildcard character. The output can be sorted based on any of the displayed columns by clicking the column name.

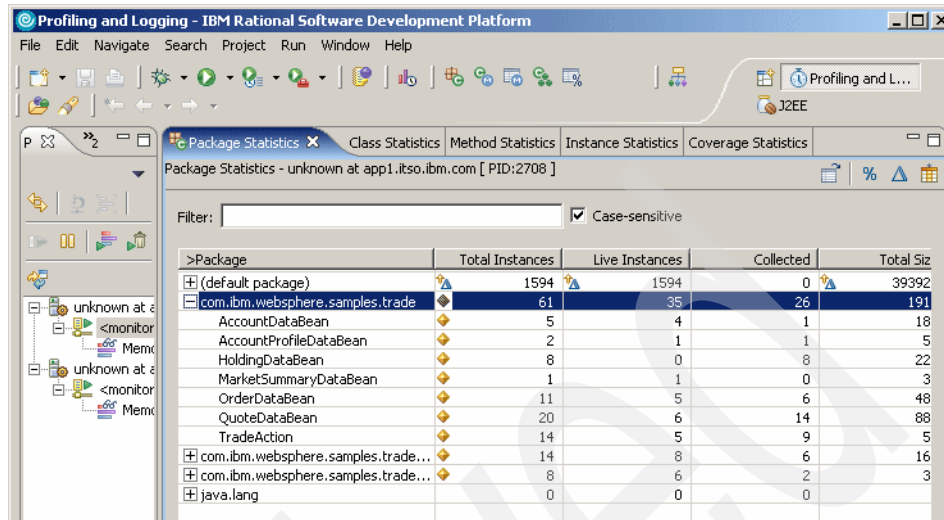


Figure 15-11 Package Statistics table

The information that is displayed is:

- ▶ Package
- ▶ Delta icon
- ▶ Total Instances
- ▶ Live Instances
- ▶ Collected
- ▶ Total Size
- ▶ Active Size
- ▶ Base Time (not displayed by default)
- ▶ Inherited Base Time (not displayed by default)
- ▶ Cumulative Time (not displayed by default)
- ▶ Inherited Cumulative Time (not displayed by default)
- ▶ Calls (not displayed by default)
- ▶ Inherited Calls (not displayed by default)

Class Statistics table

The Class Statistics table (Figure 15-12 on page 860 shows the Class Statistics when using the Java Profiling Agent) displays data for the classes with an ability to drill down to the method level.

The data in this view provides the next lower level of details, after the Package Statistics. It shows which classes are most commonly used, the execution time for methods and the size of the objects. This makes it easier to identify particular classes that require further investigation.

Profiling and Logging - IBM Rational Software Development Platform

File Edit Navigate Search Project Run Window Help

Package Statistics **Class Statistics** Method Statistics

Class Statistics - unknown at app1.itso.ibm.com [PID:668]

Filter: ☒ Case-sensitive

>Class Names	Package	Total Instances	Live Instances	Collected	Total Size	Active Size
[boolean]	(default package)	241	241	0	1652	1652
[byte]	(default package)	1002	1002	0	837288	837288
[char]	(default package)	13706	13706	0	3598152	3598152
[int]	(default package)	1177	1177	0	383564	383564
[long]	(default package)	50	50	0	936	936
[short]	(default package)	10	10	0	2136	2136
boolean	(default package)	0	0	0	0	0
byte	(default package)	0	0	0	0	0
char	(default package)	0	0	0	0	0
Class	java.lang	8	8	0	2336	2336
ConcreteQuoteEJB_73b52d85	com.ibm.websp...	50	50	0	1000	1000
DefaultMBeanRepository	mx4j.server	1	1	0	4	4
EJSLocalCMPQuoteEJB_73b...	com.ibm.websp...	2	2	0	104	104
EJSRemoteCMPQuoteEJB_7...	com.ibm.websp...	2	2	0	104	104
int	(default package)	0	0	0	0	0
long	(default package)	0	0	0	0	0
MarketSummaryDataBean	com.ibm.websp...	1	1	0	36	36
QuoteBeanCacheEntryImpl ...	com.ibm.websp...	103	103	0	7004	7004
QuoteDataBean	com.ibm.websp...	13	13	0	572	572
short	(default package)	0	0	0	0	0

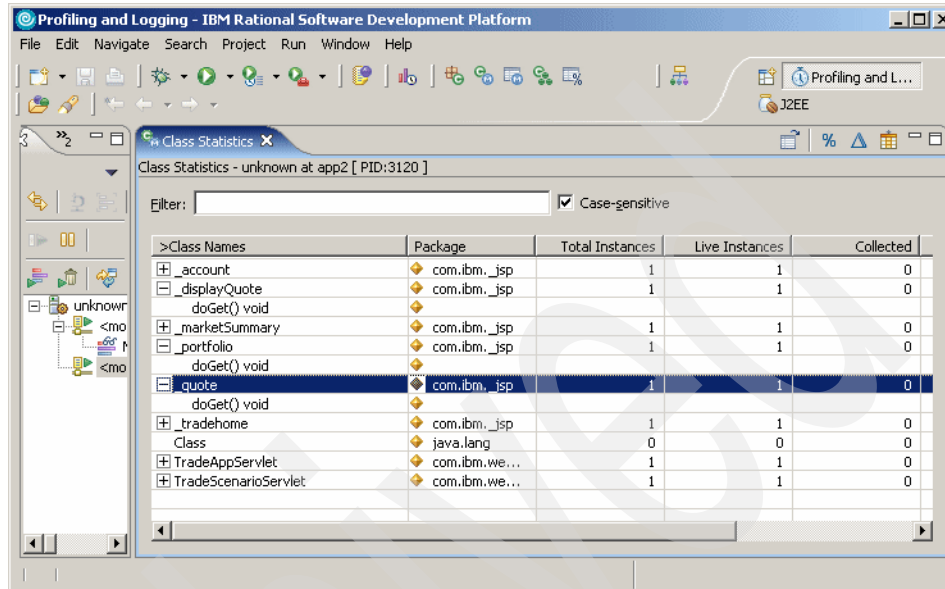
Figure 15-12 Class Statistics table - Java Profiling Agent

The information that is displayed is:

- ▶ Class Names
- ▶ Delta icon
- ▶ Package
- ▶ Total Instances
- ▶ Live Instances
- ▶ Collected
- ▶ Total Size
- ▶ Active Size
- ▶ Base Time (not displayed by default)
- ▶ Inherited Base Time (not displayed by default)
- ▶ Cumulative Time (not displayed by default)
- ▶ Inherited Cumulative Time (not displayed by default)
- ▶ Calls (not displayed by default)
- ▶ Inherited Calls (not displayed by default)

Note: Information related to instance sizes is available only if you open the view with the Java Profiling Agent.

Figure 15-13 shows the Class Statistics view when using the J2EE Request Profiler Agent:



Class Statistics - unknown at app2 [PID:3120]

Filter: ☒ Case-sensitive

>Class Names	Package	Total Instances	Live Instances	Collected
[-] _account	com.ibm._jsp	1	1	0
[-] _displayQuote	com.ibm._jsp	1	1	0
[-] doGet() void				
[-] _marketSummary	com.ibm._jsp	1	1	0
[-] _portfolio	com.ibm._jsp	1	1	0
[-] doGet() void				
[-] quote	com.ibm._jsp	1	1	0
[-] doGet() void				
[-] _tradehome	com.ibm._jsp	1	1	0
[-] Class	java.lang	0	0	0
[-] TradeAppServlet	com.ibm.we...	1	1	0
[-] TradeScenarioServlet	com.ibm.we...	1	1	0

Figure 15-13 Class Statistics table - J2EE Request Profiler

Method Statistics table

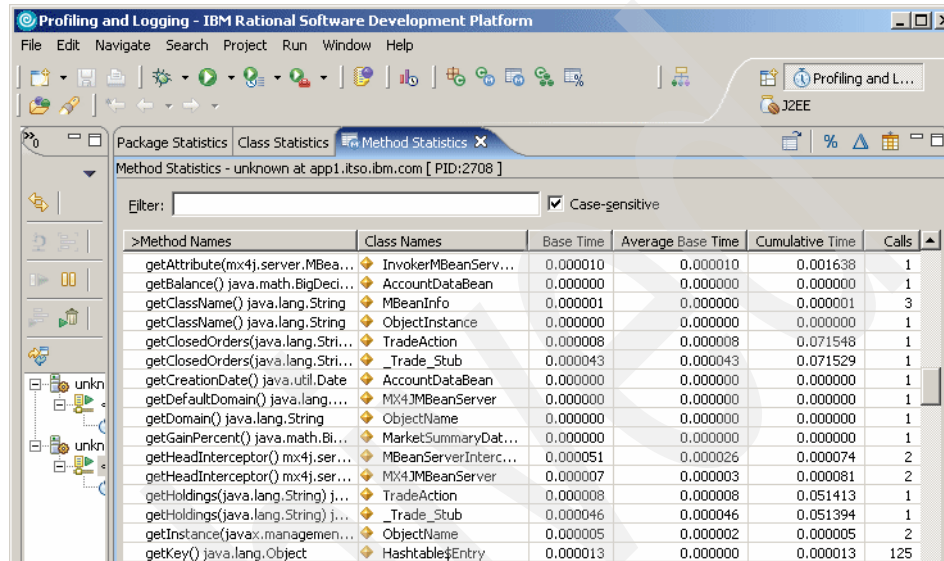
The Method Statistics table (see Figure 15-14 on page 862) allows you to view profiling data about particular methods. This table is useful in identifying the time consuming methods in an application.

It allows you to determine which particular methods are most frequently executed and those that have the longest execution time. Once these methods have been identified, effort can be focused on optimizing these methods. After making changes, the application can be profiled again and differences in performance measured. This optimization process is likely to be iterative, but the method statistics allow you to approach this in a systematic and scientific way. However, in order for the test results to be repeatable and valid comparisons made, the test environment should be the same for each test. We suggest that all other applications be closed, and that each individual test be run multiple times to reduce the probability of getting spurious results.

It contains the following information:

- ▶ Method Names
- ▶ Class Names
- ▶ Delta icon

- ▶ Package (not displayed by default)
- ▶ Base Time
- ▶ Average Base Time
- ▶ Cumulative Time
- ▶ Calls



The screenshot shows the 'Profiling and Logging - IBM Rational Software Development Platform' window. The 'Method Statistics' tab is active, displaying a table of method performance data. The table has columns for Method Names, Class Names, Base Time, Average Base Time, Cumulative Time, and Calls. The data is filtered by 'Case-sensitive'.

Method Names	Class Names	Base Time	Average Base Time	Cumulative Time	Calls
getAttribute(mx4j.server.MBean...	InvokerMBeanServ...	0.000010	0.000010	0.001638	1
getBalance() java.math.BigDeci...	AccountDataBean	0.000000	0.000000	0.000000	1
getClassName() java.lang.String	MBeanInfo	0.000001	0.000000	0.000001	3
getClassName() java.lang.String	ObjectInstance	0.000000	0.000000	0.000000	1
getClosedOrders(java.lang.Stri...	TradeAction	0.000008	0.000008	0.071548	1
getClosedOrders(java.lang.Stri...	_Trade_Stub	0.000043	0.000043	0.071529	1
getCreationDate() java.util.Date	AccountDataBean	0.000000	0.000000	0.000000	1
getDefaultDomain() java.lang....	MX4JMBeanServer	0.000000	0.000000	0.000000	1
getDomain() java.lang.String	ObjectName	0.000000	0.000000	0.000000	1
getGainPercent() java.math.Bi...	MarketSummaryDat...	0.000000	0.000000	0.000000	1
getHeadInterceptor() mx4j.ser...	MBeanServerInterc...	0.000051	0.000026	0.000074	2
getHeadInterceptor() mx4j.ser...	MX4JMBeanServer	0.000007	0.000003	0.000081	2
getHoldings(java.lang.String) j...	TradeAction	0.000008	0.000008	0.051413	1
getHoldings(java.lang.String) j...	_Trade_Stub	0.000046	0.000046	0.051394	1
getInstance(javax.managemen...	ObjectName	0.000005	0.000002	0.000005	2
getKey() java.lang.Object	Hashtable\$Entry	0.000013	0.000000	0.000013	125

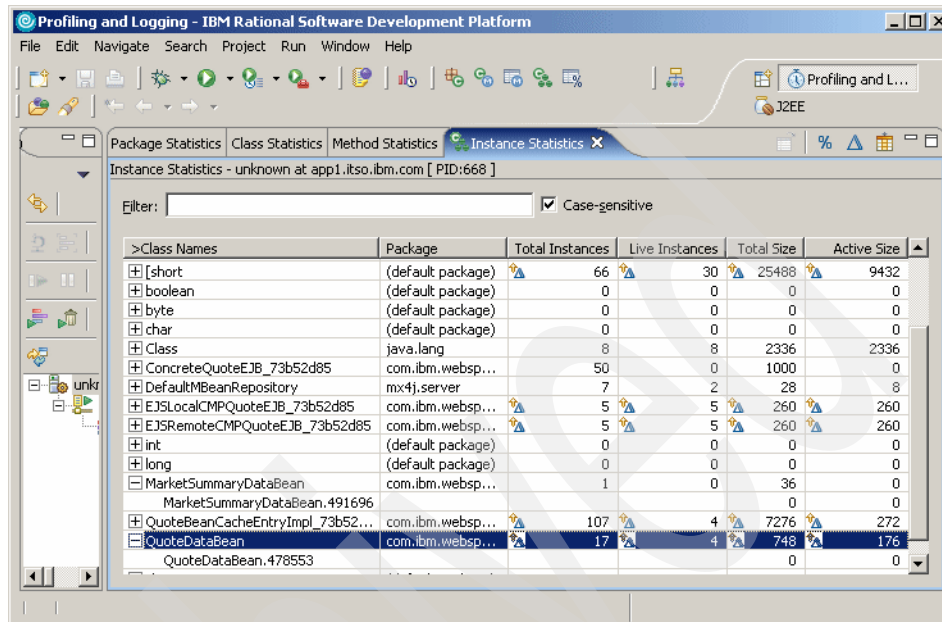
Figure 15-14 Method Statistics table

Instance Statistics Table

The Instance Statistics Table allows you to view data about particular instances of classes. The Instance Statistics Table contains the following information:

- ▶ Class Names
- ▶ Package
- ▶ Delta icon
- ▶ Total Instances
- ▶ Live Instances
- ▶ Collected (not displayed by default)
- ▶ Total Size
- ▶ Active Size
- ▶ Base Time (not displayed by default)
- ▶ Inherited Base Time (not displayed by default)
- ▶ Cumulative Time (not displayed by default)
- ▶ Inherited Cumulative Time (not displayed by default)
- ▶ Calls (not displayed by default)
- ▶ Inherited Calls (not displayed by default)

Figure 15-15 shows the Instance Statistics using the Java Profiling Agent.



Profiling and Logging - IBM Rational Software Development Platform

File Edit Navigate Search Project Run Window Help

Package Statistics Class Statistics Method Statistics **Instance Statistics**

Instance Statistics - unknown at app1.itso.ibm.com [PID:668]

Filter: ☒ Case-sensitive

>Class Names	Package	Total Instances	Live Instances	Total Size	Active Size
[short	(default package)	66	30	25488	9432
boolean	(default package)	0	0	0	0
byte	(default package)	0	0	0	0
char	(default package)	0	0	0	0
Class	java.lang	8	8	2336	2336
ConcreteQuoteEJB_73b52d85	com.ibm.websp...	50	0	1000	0
DefaultMBeanRepository	mx4j.server	7	2	28	8
EJSLocalCMPQuoteEJB_73b52d85	com.ibm.websp...	5	5	260	260
EJSRemoteCMPQuoteEJB_73b52d85	com.ibm.websp...	5	5	260	260
int	(default package)	0	0	0	0
long	(default package)	0	0	0	0
MarketSummaryDataBean	com.ibm.websp...	1	0	36	0
MarketSummaryDataBean.491696				0	0
QuoteBeanCacheEntryImpl_73b52...	com.ibm.websp...	107	4	7276	272
QuoteDataBean	com.ibm.websp...	17	4	748	176
QuoteDataBean.478553				0	0

Figure 15-15 Instance Statistics table - Java Profiling Agent

In this view, the Collected column displays the number of instances of the selected package, class, or method, that were removed during garbage collection.

From this table, it is possible to drill down to individual instances of particular classes by clicking the + symbol displayed on the left of the Class Name. By drilling down to individual instances, problems that may not be apparent just by looking at average or total values may be identified.

For example, particular instances of classes may be abnormally large. If this occurs, more details can be found by looking at the object references for the instance. To display this, right-click the instance and select **Show Object References** from the menu.

In general, determining why an object (including referenced objects) in Java requires a large amount of memory can be difficult, because the size of the object itself may be small, but there may be a set of references that eventually lead to a large object. This is a powerful tool to aid in identifying the root cause of excessive memory usage. As discussed in 16.6.1, “Memory” on page 930, minimizing memory usage is one of the key mechanisms for improving the performance of a Java application.

Tips:

- ▶ You can choose to view the references *to* other objects or *by* other objects.
- ▶ You must take a dump of the heap by running **Collect Object References** from the context menu of the active process before you display the information in the Object References Graph. IBM Rational Application Developer V6 can also automatically create heap dumps for you, just choose the **Memory Leak Analysis** profiling set (see Figure 15-5 on page 853).
- ▶ You can identify memory leaks in a particular unit of work by taking a dump just before triggering your transaction and then again at the end of the transaction. The objects that could not be collected between the two dumps will be labelled as new objects.

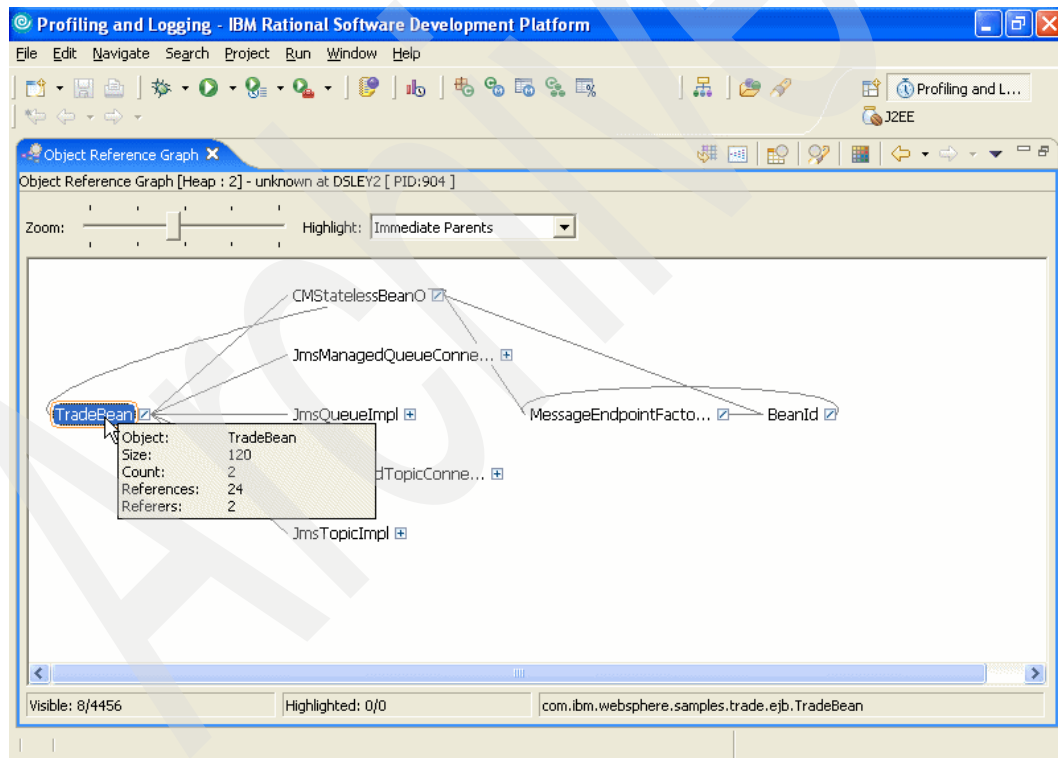


Figure 15-17 Object Reference Graph

The Object Details view

The Object Details view displays detailed information about a specific object shown in the Object Reference Graph. To open the Object Details view, double-click a node in the Object Reference Graph, or right-click a node and select **Show Object Details** from the pop-up menu.

The Object Details view consists of the following data display areas:

- ▶ **Object Details:** Information about the currently selected object.
- ▶ **Highlighted Objects list:** A list of objects that are currently highlighted in the Object Reference Graph view. Double-click a list item to navigate to the object in the Object Reference Graph view.
- ▶ **Referer table:** A list of immediate objects that reference the currently selected object, and information about each referer.
- ▶ **Referee table:** A list of immediate objects referenced by the currently selected object, and information about each referee.

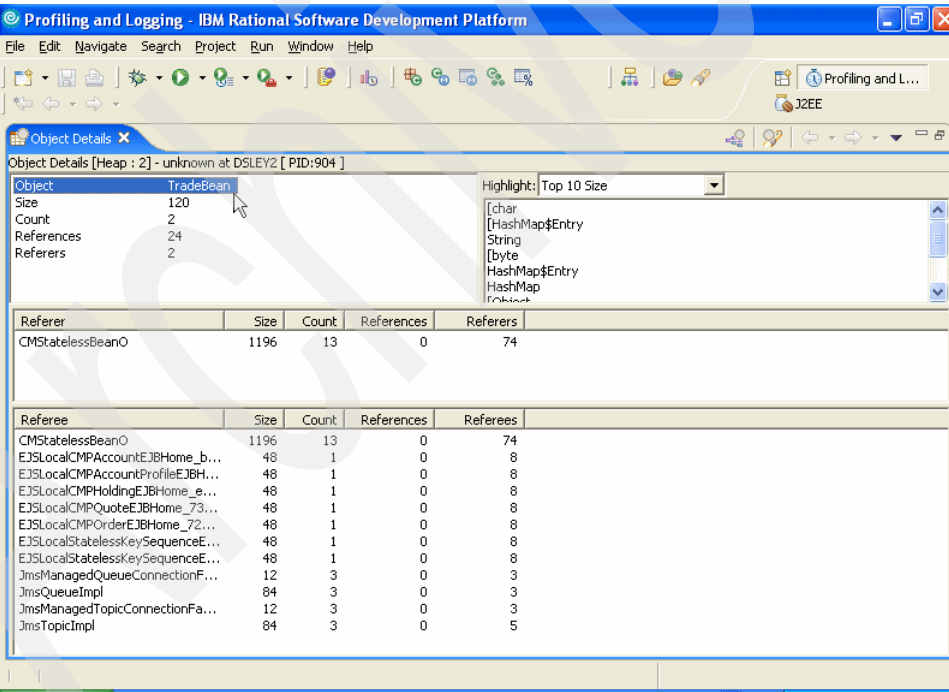


Figure 15-18 Object details for the TradeBean

Execution Flow view

The Execution Flow view visually displays the method calls performed by the application being profiled. In order to reduce the amount of information being displayed to a manageable level, we recommend that you use the J2EE Request Profiler Agent for capturing data to be viewed in the Execution Flow view. The Execution Flow view is another way of determining which methods are executed most frequently, and hence are good candidates for optimization. An example of the Execution Flow view is provided in Figure 15-19.

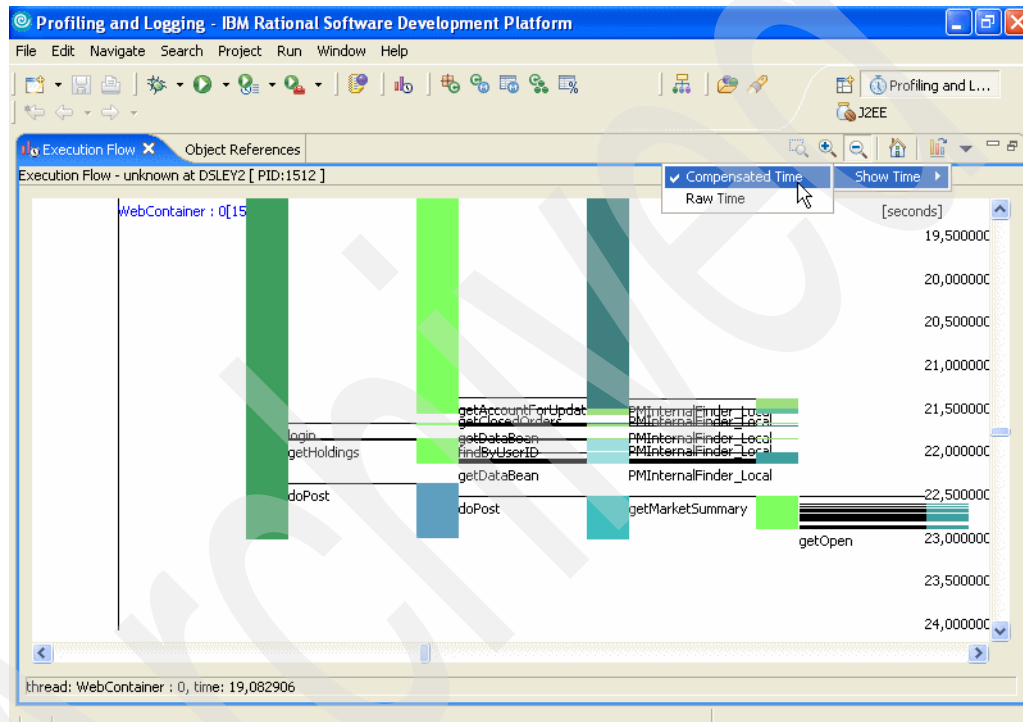


Figure 15-19 Execution Flow view

The information that is displayed pictorially in the Execution Flow view can also be displayed in a hierarchical tabular form as shown in Figure 15-20 on page 868. In some cases it may be easier to navigate through the data in this view, because it is possible to expand and collapse particular parts of the execution sequence. From this view, you can right-click a particular method invocation to display the Method Invocation view for that method.

Profiling and Logging - IBM Rational Software Development Platform

File Edit Navigate Search Project Run Window Help

Execution Flow Object References **Execution Flow Table**

Execution Flow Table - unknown at DSLEY2 [PID:4188]

Filter: ☒ Case-sensitive

Thread Names	Instance Name	>Start Time	Cumulative Time
WebContainer : 0[4188]			
doGet	TradeAppServlet.3	11,350000	0,181000
doGet	_welcome.8	11,511000	0,020000
doPost	TradeAppServlet.3	13,317000	2,619000
getClosedOrders	TradeBean.16	13,789000	1,676000
login	TradeBean.16	15,505000	0,010000
findByPrimaryKeyForUpdate	ConcreteAccountProfileEJB_3bd34e...	15,505000	0,000000
getAccountForUpdate	ConcreteAccountProfileEJB_3bd34e...	15,505000	0,000000
login	ConcreteAccountEJB_b5483e03.52	15,515000	0,000000
getDataBean	ConcreteAccountEJB_b5483e03.52	15,515000	0,000000
getAccountData	TradeBean.16	15,585000	0,010000
getHoldings	TradeBean.16	15,595000	0,120000
doPost	_tradehome.162	15,796000	0,140000
doGet	TradeAppServlet.3	21,195000	0,181000
doGet	TradeAppServlet.3	25,811000	0,211000
doGet	TradeAppServlet.3	31,120000	0,913000
doGet	TradeAppServlet.3	64,920000	0,231000
doGet	TradeAppServlet.3	68,573000	0,150000
doGet	TradeAppServlet.3	71,814000	0,061000
doGet	TradeAppServlet.3	77,374000	0,271000
WebContainer : 1[4188]			
Default : 0[4188]			

Figure 15-20 Execution Flow table

Object and Class Interaction (Sequence Diagram) views

The Object Interaction and Class Interaction views (also labeled as UML2 Sequence Diagrams) allow analysis for program execution by displaying UML (Unified Modelling Language) sequence diagrams.

Normally these diagrams are produced statically during the analysis or design phase of a project, but IBM Rational Application Developer V6 allows you to dynamically generate these diagrams to see how the program is really working, rather than how it *should* be working.

These views provide another means of analyzing the execution of your application, helping to focus your optimization efforts. The difference between the Object and Class Interaction views is that the Object Interaction view displays each individual instance of each class separately. An example of the Class Interaction view can be seen in Figure 15-21 on page 869.

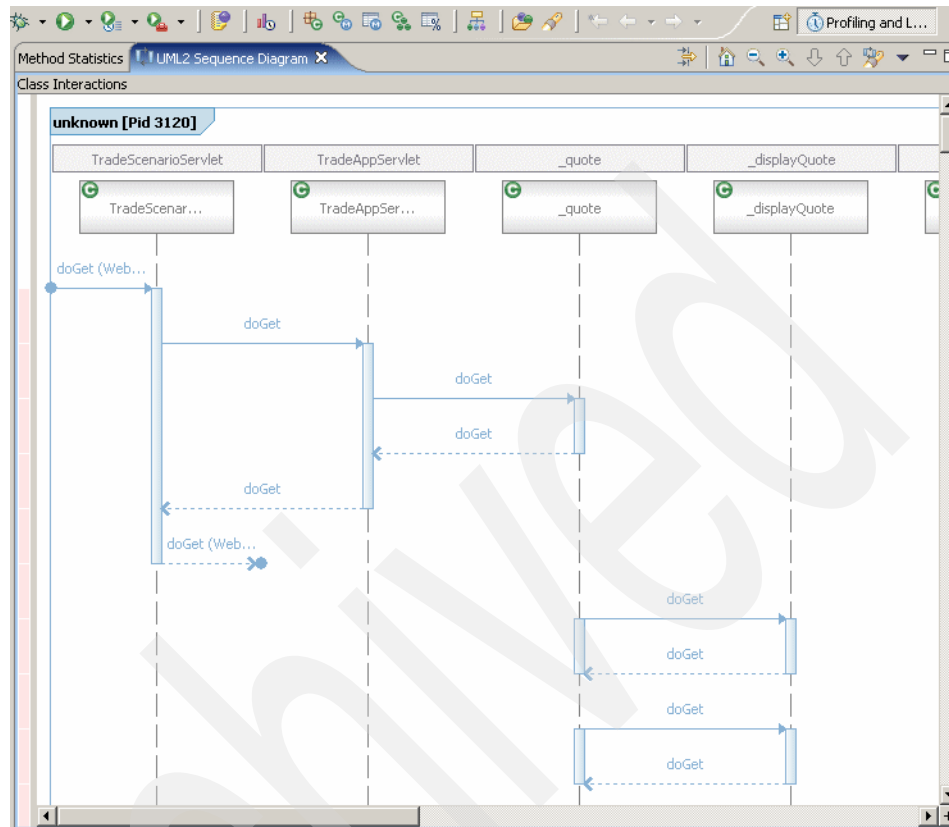


Figure 15-21 Class Interaction view

You can view the sequence of the execution flow from different levels of abstraction, starting with class interactions, through interactions among threads, or process interactions, up to hosts interactions across a network. The importance of this multilevel data presentation becomes obvious for the monitoring of e-business applications.

Depending on the application infrastructure, you may need to view the execution flow on different levels of the monitoring hierarchy. For a local application, the level of process or host may be adequate, but for a distributed application, the monitor level, which provides a view of the execution across multiple hosts, could be more appropriate. Viewing the execution of a distributed application from the monitor level may reveal some points of interest which can direct application developers to any of the lower level graphs to perform more detailed viewing or analysis. For example, in the case of an application consisting of servlets and enterprise beans that are distributed across a cluster of hosts, the preliminary view of the host interactions may lead a software developer to view sequence

diagrams of object interactions for specific processes. These diagrams will show only selected data, representing part of the execution within an enterprise bean or servlet container, on the level of corresponding business components.

To navigate the data acquisition hierarchy, you can use the Profiling Monitor view. Each of the hierarchy levels of the profiling resources provides all the applicable types of sequence diagrams.

The following types of diagrams are available:

- ▶ **Class interactions**
Class interaction diagrams can be used to view interactions of class methods that participate in the execution of an application.
- ▶ **Object interactions**
Object interaction diagrams can be used to view interactions of object methods that participate in the execution of an application.
- ▶ **Thread interactions**
Thread interaction diagrams can be used to view interactions of methods that execute in different threads, which participate in the execution of an application. See “Thread Interactions view” on page 872 for more information.
- ▶ **Process interactions**
Process interaction diagrams can be used to view interactions of methods that execute in different processes, which participate in the execution of an application. “Process Interaction view” on page 871 gives more details.
- ▶ **Host interactions**
Host interaction diagrams can be used to view interactions among methods that execute on different hosts, which participate in the execution of a distributed application. Host interaction diagrams provide the highest level of abstraction in a sequence diagram presentation. The flow of interactions presents the execution of these methods across machines. See Host Interaction view below for more information.

Host Interaction view

The Host Interaction view is the highest possible level of diagram available. Use this diagram to view interactions among methods that run on different hosts.

Important: The default filters for the Profiling Monitor perspective have to be changed, otherwise the UML2 Host Interactions view cannot be selected. To do this, disable the **Hide Monitors** filter by selecting ▼ in the upper right corner of the Profiling Monitor pane.

The Host Interaction view opens from the context menu of the DefaultMonitor resource as shown in Figure 15-22.

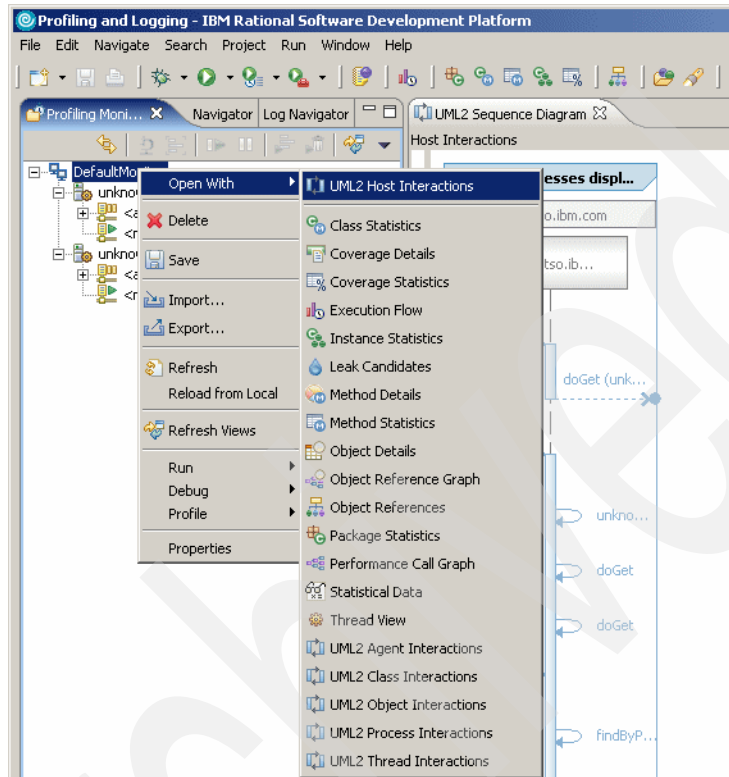


Figure 15-22 Open Host Interaction view

Process Interaction view

Use the Process Interaction diagram to view the interactions among methods that run in different processes of the same host. This view opens from the context menu of the host as shown in Figure 15-23 on page 872.

Note: To select the **Process Interaction** view, the default filters also need to be changed. Disable the **Hide Hosts** filter as explained in the previous Note.

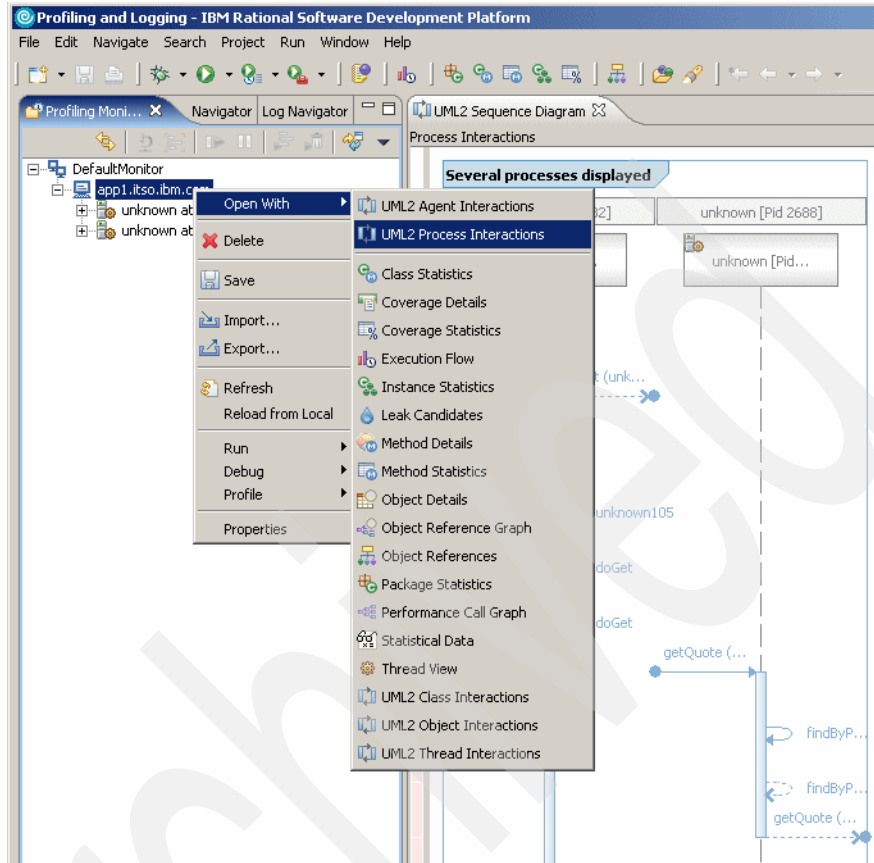


Figure 15-23 Open Process Interaction view

Thread Interactions view

Use the Thread Interaction diagram to view the interactions among methods that execute in different threads of the same process. This view opens from the context menu of the resource that represents a monitor, host, or a process. See Figure 15-24 on page 873.

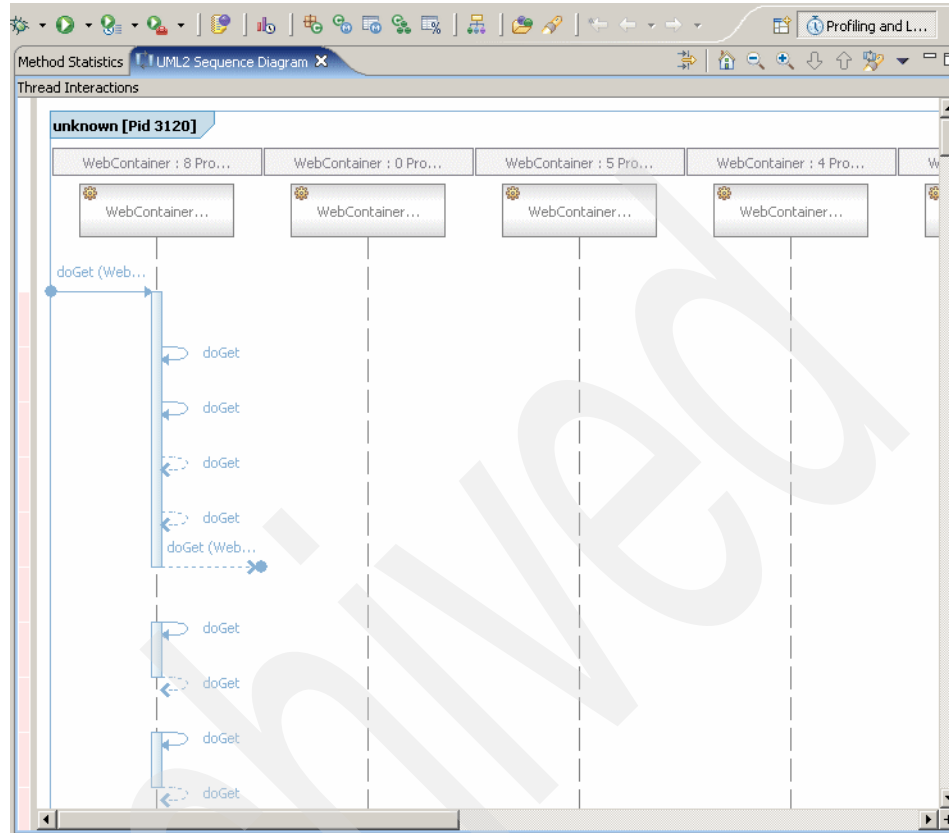


Figure 15-24 Thread Interactions view

15.2.7 Resolving performance bottlenecks - Execution time analysis

Profiling gives you insight into the performance characteristics of an application, and its runtime behavior. Both helps you to get a better understanding of an applications' resource requirements and finding potential performance bottlenecks.

This section explains how to use the Profiler to analyze the performance of a WebSphere based J2EE application. The different steps needed to perform a runtime analysis and to detect and resolve a performance bottleneck are shown using a modified Trade 6 sample application. We provide an example of how to use the Execution time analysis. If you are interested in an example of how to use the code coverage analysis, see Chapter 24, "Profile applications" in *Rational Application Developer V6 Programming Guide*, SG24-6449.

Note: If you want to follow the steps explained here, we recommend that you first read the entire section, then import the Trade 6 application into your workbench, make the appropriate changes to the source code as shown in Example 15-1, then start with the testing.

Setting up the environment

Before you can start analyzing the performance of your application, some prerequisites must be satisfied. First it is not recommended that you run the tested application and IBM Rational Application Developer V6 on the same machine because of the side effects when using shared hardware. Therefore, running performance tests in the Rational Application Developer's test environment is also not an option. In the following scenario, two separate machines are used, one client machine with all necessary test tools and a separate WebSphere Application Server machine with the Trade 6 application.

Client machine prerequisites

- ▶ IBM Rational Application Developer V6 installed.
- ▶ Either a load testing tool or a test client (for example a browser for Web-based applications).
- ▶ Availability of an automated test script or a step-by-step description of the test scenario.

WebSphere Application Server machine prerequisites

- ▶ IBM Rational Agent Controller must be installed and should be running.
- ▶ Profiling on the WebSphere Application Server must be enabled (see "Enable Profiling in WebSphere Application Server" on page 848).
- ▶ Application to be analyzed, in this example the modified Trade 6 application. Please verify, that there are no other applications running neither on the application server nor directly on the WebSphere system.

Connecting Rational Application Developer to WebSphere Application Server

After verifying the test environment, we are now ready to start the performance analysis of our sample application. In this part the steps to configure and establish the connection for profiling the Trade 6 application on the remote WebSphere Application Server are explained.

1. Start Rational Application Developer with an empty workspace.
2. Open the Profiling Perspective. Select **Window -> Open Perspective -> Other**.

In the Select Perspective window, first select the **Show all** checkbox to make the **Profiling and Logging** view available for selection.

3. Select **Profiling and Logging** and click **OK**. Answer **OK** to the following question regarding enablement of Profiling and Logging.
4. Now a suitable profiling configuration has to be created:
 - a. Select **Run -> Profile** to open the profile management and configuration window.
 - b. To profile a remote WebSphere Application Server V6, select **Attach - Java Process**, make sure that the **Profiling and Logging** profile is selected in the dropdown list in the right pane and click **New**. Alternatively you can double-click **Attach - Java Process**.
 - c. Change the Name of the configuration to **WAS6_Remote_Cfg**, delete the existing default host **localhost** and define the remote system you wish to profile (and which has the IBM Agent Controller installed and running) as the new Default Host. **Test** the connection, then select **Apply** to save the changes. Figure 15-3 on page 851 is a reference for this task.
5. Switch to the **Agents** tab to select the available agents on the remote machine.

Make sure that the application server is started. Starting the server causes the respective embedded Profiling agents to be displayed. The Process ID (PID) values indicated here correspond to the application server's PID allocated by the operating system at server startup.

Select both the **J2EE Request Profiler** and the **Java Profiling Agent** as indicated in Figure 15-25.

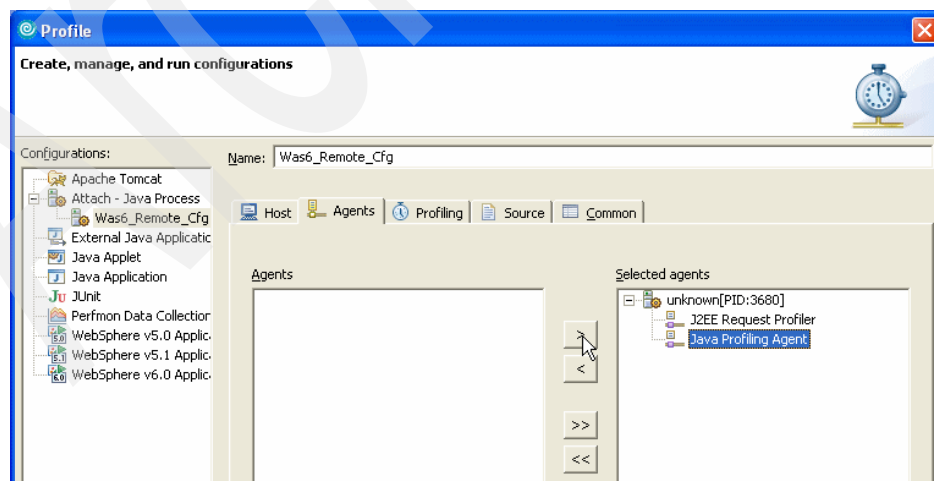


Figure 15-25 Agent configuration

6. Configure the profiling set.

The Java Profiling Agent was selected for the detailed analysis of the WebSphere Application Server based application. On the Profiling tab, the corresponding profiling sets can be selected. Select the **Profiling** tab.

Specifying profiling sets and creating profiling filters enables you to specify the subset of information that you want to analyze. Profiling sets specify the type of data to collect and filters ensure that only relevant details are channeled out to the views. Using filters is especially useful when speed and efficiency is critical: the less data there is, the less impact it has on the system, and the faster it can be collected.

- a. For our performance analysis of the modified Trade 6 application the predefined **Execution History - Full Performance Call Graph** should be selected (see Figure 15-26).

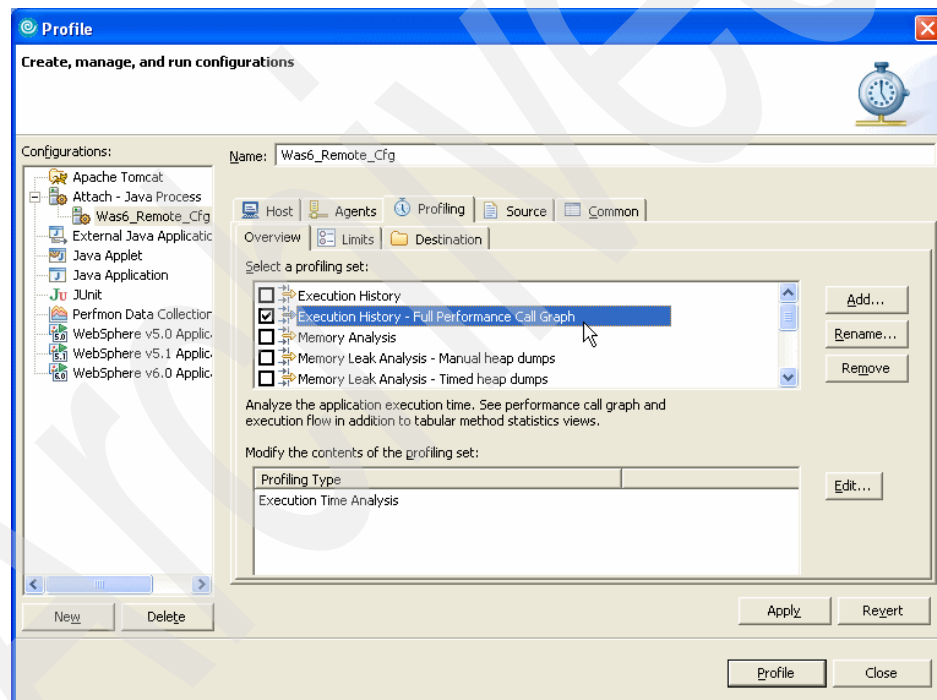


Figure 15-26 Select profiling set

- b. Next the default filters for the Execution History set have to be changed to enable profiling of our sample application. Usually it makes sense to filter out all packages that start with `com.ibm`, but our sample Trade 6 application classes also start with this name. Therefore, select **Edit** to change the Execution History profiling set. Click **Next**.

- c. On the following screen, choose the **WebSphere J2EE** filter set and **include** the package **com.ibm.websphere.samples.trade*** which is used by the application. See Figure 15-27. Click **Finish**.

Tip: Filters are applied top down. When adding new filters, make sure that the order is correct.

The other profiling options (Limits, Destination) can remain unchanged.

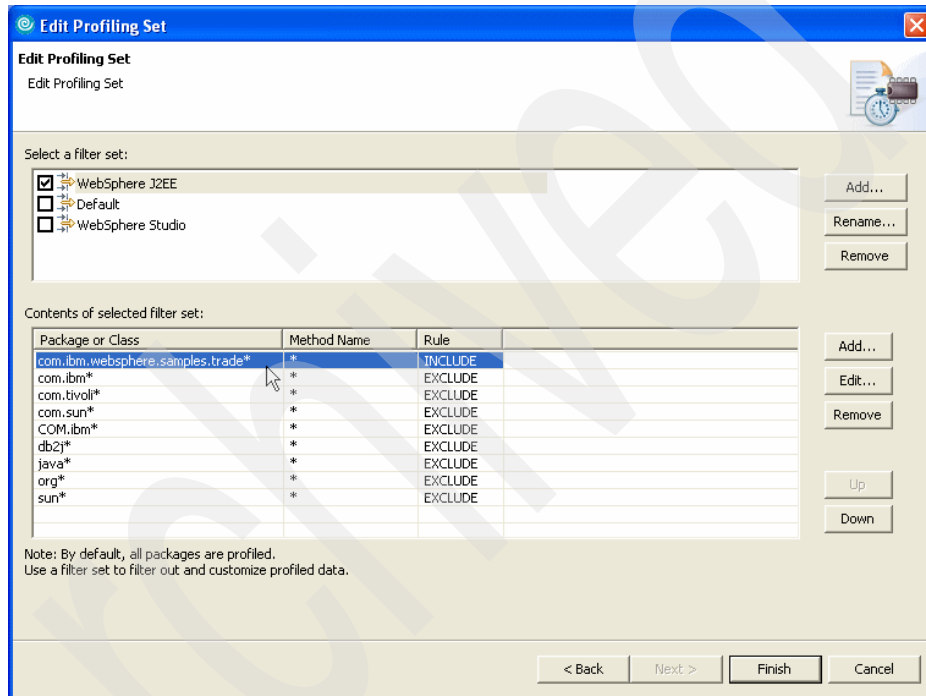


Figure 15-27 Modify existing profiling set

7. Define application sources.

If the application sources are available, it is recommended that you add them in the Source tab. This enables easy access to the sources during the performance analysis.

- First **Import** trade.ear into the workbench as a new project.
- Next attach the source code: Go to the **Source** tab, uncheck **Use default source lookup path**, then click **Add Projects...** as seen in Figure 15-28 on page 878 and select all projects (see Figure 15-29 on page 878).

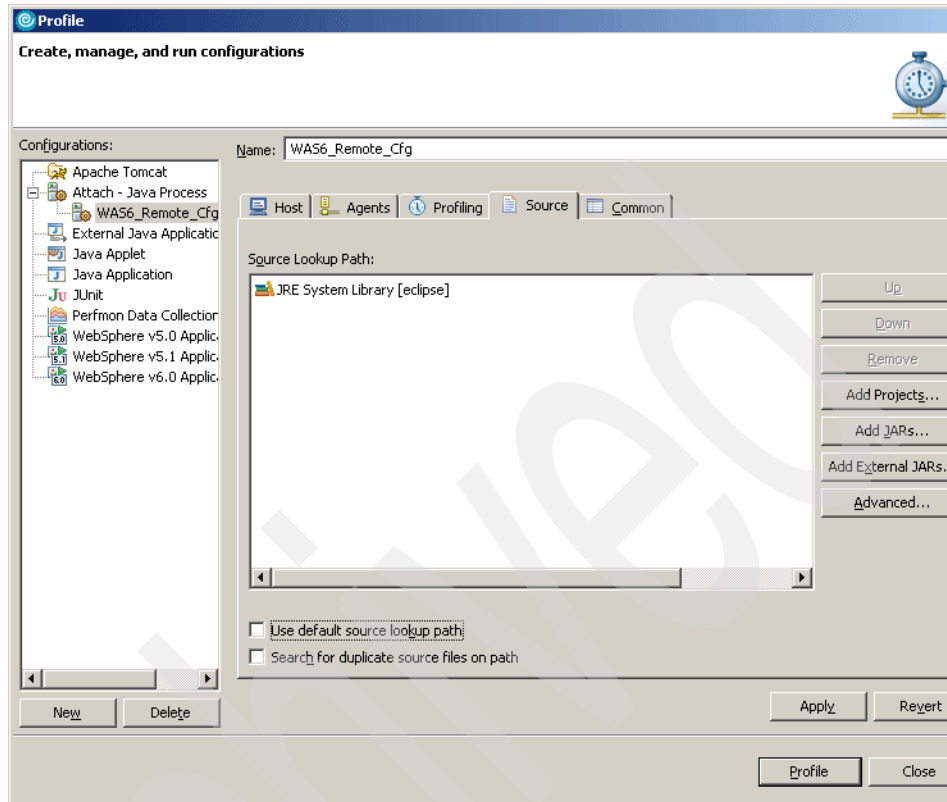


Figure 15-28 Adding a source

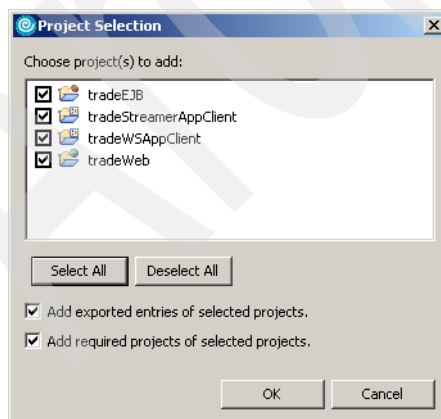


Figure 15-29 Selecting a project

8. All necessary changes to the profile configuration are completed. Click **Profile** to save the configuration and return to the Profiling perspective. Accept the following information message that reminds you to start monitoring.

Start Monitoring

After the profiling configuration is completed, the monitoring of the modified Trade 6 sample application can be started. To do this, right-click the profiling agents and select **Start Monitoring** (see Figure 15-30). Make sure that monitoring is activated for both agents - the state should change from <attached> to <monitoring>.

Now use the WebSphere application, exercising the routine you want to investigate for performance bottlenecks. The monitors collect and record the performance data instantly.

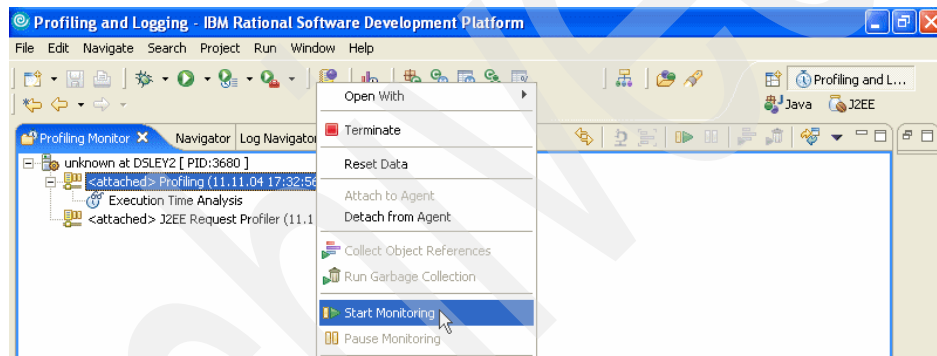


Figure 15-30 Start Monitoring

Test scenario

While the IBM Rational Application Developer V6 profiling tools are running, the following test scenario should be executed:

1. Load the Trade 6 Homepage: <http://app1:9080/trade/>
2. Select the **Go Trade!** link.
3. **Log in** (using default user).
4. Select the **Account** link.
5. Change to the **Portfolio** page.
6. **Sell** one symbol (randomly selected).
7. Go to **Quotes/Trade**.
8. **Buy** one symbol.
9. Logoff.

After performing the scenario above, stop monitoring in the same way as starting it (just click **Pause Monitoring** instead of Start Monitoring).

Analyzing Performance

The primary views for viewing and analyzing performance data are the Performance Call Graph and the Method Details views. You can supplement these with the Package Statistics, Class Statistics, Method Statistics, Method Invocation, Method Invocation Table, Execution Flow, Execution Flow Table, and the UML2 Sequence Diagram views.

J2EE Request Profiler

To analyze the application performance and detect performance bottlenecks, it is recommended that you start with the Performance Call Graph of the J2EE Request Profiler. It provides a high-level performance overview data about the application.

Therefore, select the **J2EE Request Profiler** and choose **Open With -> Performance Call Graph** (see Figure 15-31) from the context menu.

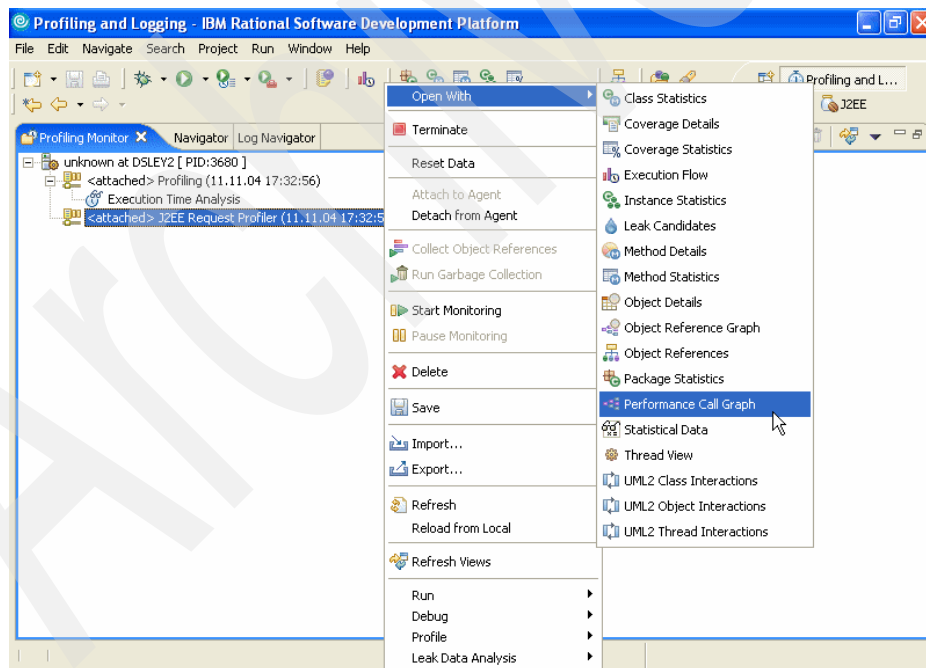


Figure 15-31 Open Performance Call Graph

For the modified Trade 6 sample, the call graph shown in Figure 15-32 on page 881 is displayed:

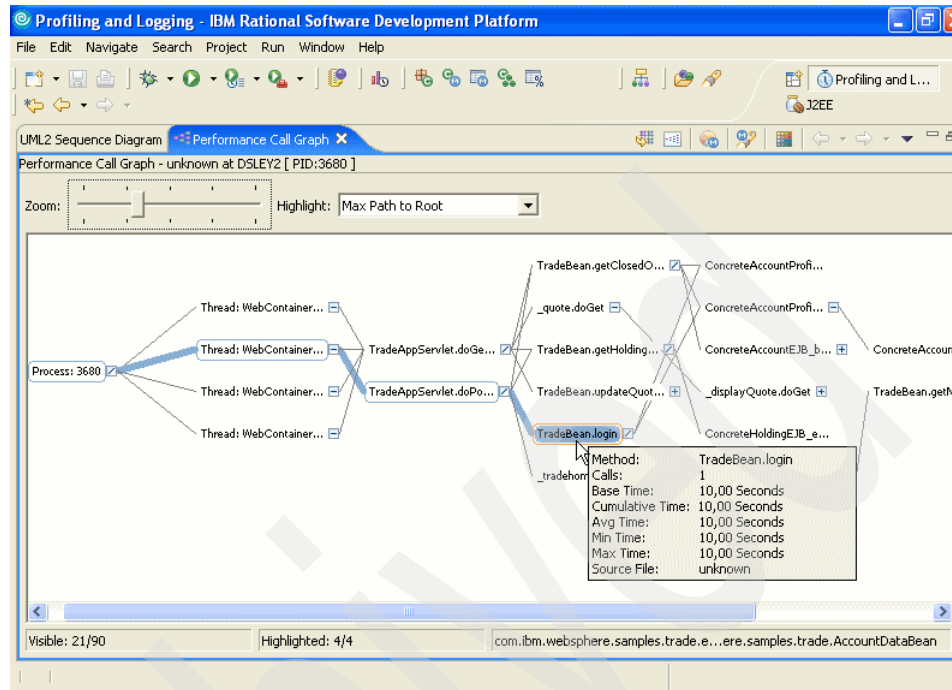


Figure 15-32 Performance Call Graph for the Trade 6 test scenario

- ▶ The graph initially displays the 20 methods that are responsible for consuming the largest amount of time as nodes.
- ▶ The lines between the nodes represent call paths. Thicker lines are used for more expensive call paths. You can focus the call graph by right-clicking a node and choosing a command from the pop-up menu.
- ▶ Identify a method that you suspect of consuming more time than it should. Double-click the method to open the Method Details view.

Use the Performance Call Graph and the Method Details view together to investigate the entire data set. The two views are synchronized whenever you select a new method.

To examine the source code for a method, right-click it in either of these views and select **Open Source** from the pop-up menu.

Once you had a detailed look at the Trade 6 performance graph shown above, it is easy to realize, that the `TradeBean.login()` method is experiencing poor execution times. To investigate the problem further, the Method Details view can be used to analyze the `TradeBean.login` method.

Right-click the `TradeBean.login()` method and select **Show Method Details** from the context menu. The following details are available for the `TradeBean.login()` method (see Figure 15-33):

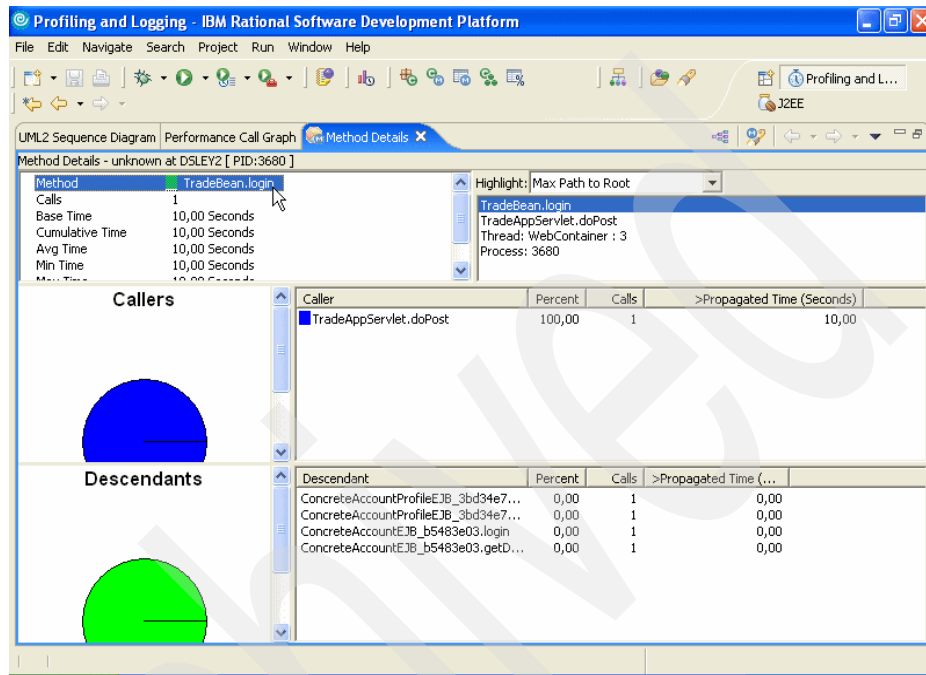


Figure 15-33 Method Details for `TradeBean.login()`

The Method Details view displays data about a specific method and its immediate callers and descendants.

The Method Details view consists of the following data display areas:

- ▶ **Method Details:** Information about the currently selected method.
- ▶ **Highlighted Methods list:** A list of methods that are currently highlighted in the Performance Call Graph. Double-click a list item to display the method in the Method Details view.
- ▶ **Caller table:** A list of immediate methods that call the method that is currently displayed, and information about each caller.
- ▶ **Callers pie chart:** A chart showing the percentage of the method's cumulative time that is attributed to invocations by each method in the Callers table.
- ▶ **Descendant table:** A list of immediate methods that are called by the method, and information about each descendant.

- **Descendants pie chart:** A chart showing the percentage of the method's cumulative time that is attributed to the cumulative time of each method in the Descendants table.

When looking at the method descendants in detail, it is obvious that the `TradeBean.login()` method consumes about 100% of the time that is needed for the whole call.

So what is going on in this method?

One solution to figure this out, is, of course to look into the sources. To do this, right-click the method and select **Open Source** from the context menu. But keep in mind that until now just the J2EE Request Profiler was used. The J2EE Request Profiler only displays J2EE Components like Enterprise JavaBeans, servlets, and so on. All other calls to “pure” java objects are filtered out. This makes sense, because the intention of the J2EE Request Profiler is to provide an overview and not to go into all the details.

Java Profiling Agent

To get a more detailed insight on what is going on in this method and to find out if other java objects are called that consume a huge amount of execution time, the Java Profiling Agent is used.

To open the more detailed Performance Call Graph from the Java Profiling Agent, right-click the agent in the Profiling Monitor and select **Open With -> Performance Call Graph**. The java profiling Performance Call Graph for the modified Trade 6 application unveils the reason for the long execution time - there is a `sleep()` method called! See Figure 15-34.

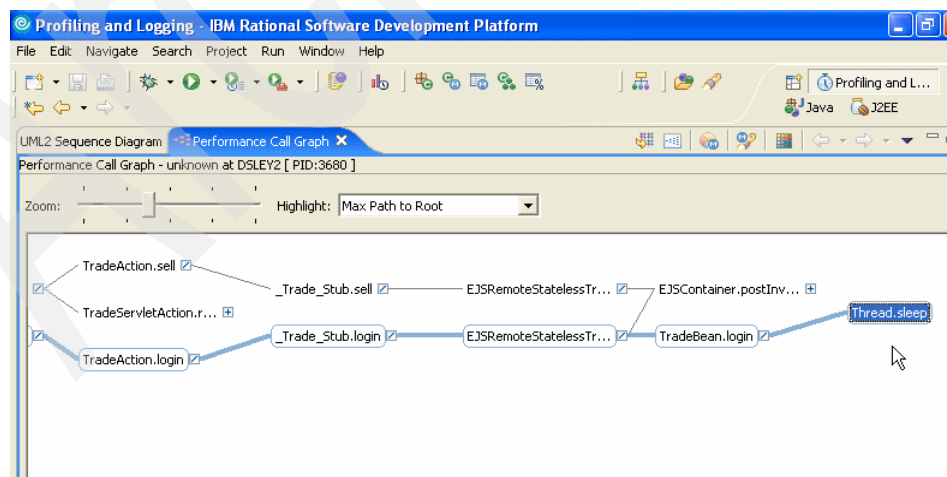


Figure 15-34 Performance Call Graph from the Java Profiling agent

Now let's have a look at Example 15-1 which shows the source of the login() method.

Example 15-1 login() method

```
public AccountDataBean login(String userID, String password)
    throws FinderException, Exception {
    LocalAccount account = ((LocalAccountProfile)
        profileHome.findByPrimaryKeyForUpdate(userID)).getAccountForUpdate();

    if (Log.doTrace())
        Log.trace("TradeBean:login", userID, password);
    account.login(password);
    Thread.currentThread().sleep(10000);
    AccountDataBean accountData = account.getDataBean();
    if (Log.doTrace())
        Log.trace("TradeBean:login(" + userID + ") success" + accountData);
    return accountData;
}
```

Resolve performance bottleneck

Remove the sleep() statement from the modified Trade 6 application, then verify that the cumulative time for the TradeBean.login() method has been reduced and this performance bottleneck has been resolved. One way to do this is to repeat the test scenario. This way, you should see whether the login() method still consumes as much time in the Performance Call Graph; it will not.

However, to demonstrate the capability of the Profiler, we are using a different approach. Install the updated Trade 6 application on your WebSphere Application Server and repeat the previous monitoring steps.


Verify that performance bottleneck is resolved

To analyze the data, the UML2 Object Interactions sequence diagram of the J2EE Request Profiler is used.

Open the Object Interactions sequence diagram. Right-click the **J2EE Request Profiler** agent and select **Open With -> UML2 Object Interactions**.

The Sequence Diagrams are used to find the poorly performing spots in the application. To find these spots, look for the red squares in the bar on the left side of the Sequence Diagram view. The intensity of the red color of the rectangles corresponds to the amount of time elapsed between consecutive events in the diagram. This means that the darker red a square is, the more time was consumed by the particular event(s) at that point in the diagram.

Tip: If the red squares don't show up in your sequence diagram, use the Zoom in the diagram smarticon in the upper right corner of the diagram.

Use the search functionality of the sequence diagram views to find the `TradeBean.login()` method. Select the Find smarticon  in the upper right corner of the Sequence Diagram view. In the Search window, enter ***login*** into the Matching string field, select the **Synchronous message** checkbox for the synchronous method call, and click **Find**.

The `TradeBean.login()` method will be highlighted in the sequence diagram (see Figure 15-35).

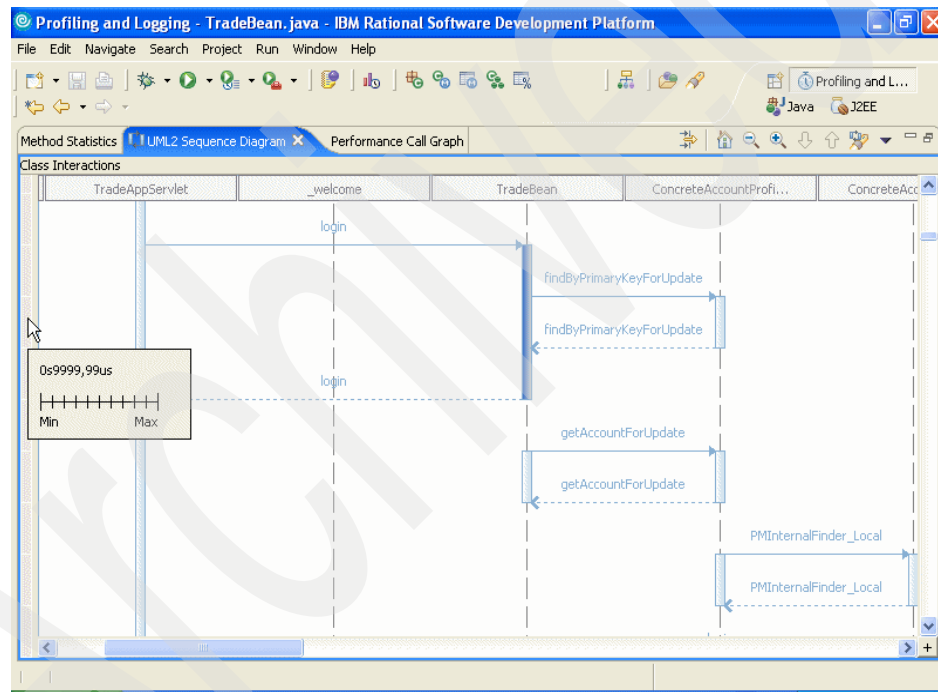


Figure 15-35 UML Sequence Diagram for Trade 6

To verify the amount of time spent in the login method, move the mouse cursor over the light red squares in the bar on the left side of the Sequence Diagram view. It is definitely less than before - and the light red square also indicates that this performance bottleneck has been resolved successfully.

15.3 IBM Page Detailer

IBM Page Detailer is a browser-side tool to measure performance of a Web application. While the Profiler discussed in the previous sections supports analysis of the execution of the application on the server, the Page Detailer collects most of its useful data at the socket level to reveal the performance details of items in the Web page, from the client's (browser's) perspective. It is also useful for measuring the incremental impact of changes in a Web application.

Page Detailer allows you to look at how and when each item is loaded in a Web page. Analyzing this data allows you to identify the areas where performance could be improved. The user's perception of performance is determined based on the time to display pages, so measuring and analyzing this data will provide insight into the user's experience of your application.

15.3.1 Overview

IBM Page Detailer is a graphical tool that enables Web site developers and editors to rapidly and accurately assess performance from the client's perspective. IBM Page Detailer provides details about the manner in which Web pages are delivered to Web browsers. These details include the timing, size, and identity of each item in a page. This information can help Web developers, designers, site operators and IT specialists to isolate problems and improve performance and user satisfaction. Page Detailer can be used with any site that your browser can access.

Page Detailer is a separately downloadable product that can be obtained from IBM alphaWorks at:

<http://www.alphaworks.ibm.com/tech/pagedetailer>

There are two versions available:

- ▶ **Evaluation version:** This version is for free but does not support all features.
- ▶ **Pro version:** This version must be licensed for a small fee and contains the following additional features:
 - Full support for HTTPS (SSL) traffic
 - Saving and restoration of captured data
 - Ability to add/edit one's own notes for captured pages and items
 - A find facility for working with text

The supported platforms for Page Detailer are Windows 2000 and Windows XP.

The Page Detailer can monitor all HTTP and HTTPS requests originating from the machine where it is running. Thus it can be used to measure performance for

Web applications running either locally or remotely, on either WebSphere Application Server or IBM Rational Application Developer V6.0 or any other Web site or application. Hence the Page Detailer may be used at any time during the project development cycle, from coding through to production support. Note that when analyzing the performance of non-production environments, differences in the production environment topology and configuration could result in differences in the measured performance results.

IBM Page Detailer gathers the following information:

- ▶ Connection time
- ▶ Socks connection time and size
- ▶ SSL connection time and size
- ▶ Server response time and size
- ▶ Content delivery time and size
- ▶ Delays between transfers
- ▶ Request headers
- ▶ Post data
- ▶ Reply headers
- ▶ Content data
- ▶ Page totals, averages, minimums, and maximums

For each page that is accessed, a color-coded bar chart of the time taken to load the page items will be generated. The length of a particular bar gives a good idea of the relative time spent in loading that item, as compared to the whole page. You will see that in some cases, items of a page may be loaded in parallel. This will appear in the chart with bars that overlap. The information that is captured by the Page Detailer includes page size as well as sizes of all other items loaded by the browser.

Different colors in the bar indicate how the time was spent.

- ▶ Page Time (Purple)
The time taken to load all the components of a page.
- ▶ Host Name Resolution (Cyan)
The time spent to resolve the IP address of the host.
- ▶ Connection Attempt Failed (Brown)
The time taken to receive an error when a connection attempt is made.
- ▶ Connection Setup Time (Yellow)
The time taken to open a socket connection. If a SOCKS server is being used, this is the time to open a socket connection from the browser to the SOCKS server only.

- ▶ Socks Connection Time (Red)
The time taken to open a connection from a SOCKS server to the remote site.
- ▶ SSL Connection Setup Time (Pink)
This is the time taken to negotiate an encrypted connection between the browser and the remote site, once a normal socket connection has been established.
- ▶ Server Response Time (Blue)
This is the time from the browser's request to the receipt of the initial reply, after all the communications setup has been completed. Large responses are broken down into smaller components (packets). The server response time only measures the time to receive the first one.
- ▶ Delivery Time (Green)
The time taken to receive all additional data that was not included in the initial response.

An example of a chart produced with Page Detailer is provided in Figure 15-36.

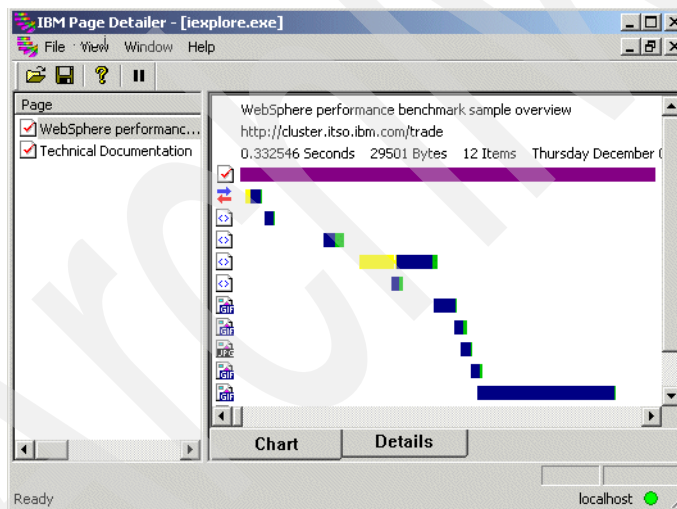


Figure 15-36 Page Detailer Chart view

To obtain more detailed information about a particular HTTP request, double-click the appropriate colored bar or the icon in the chart. This will display a text viewer as shown in Figure 15-37 on page 889. It includes information about timings, sizes, header, etc. You can add your own notes and save the file for comparison. This feature is available only in the Pro version.

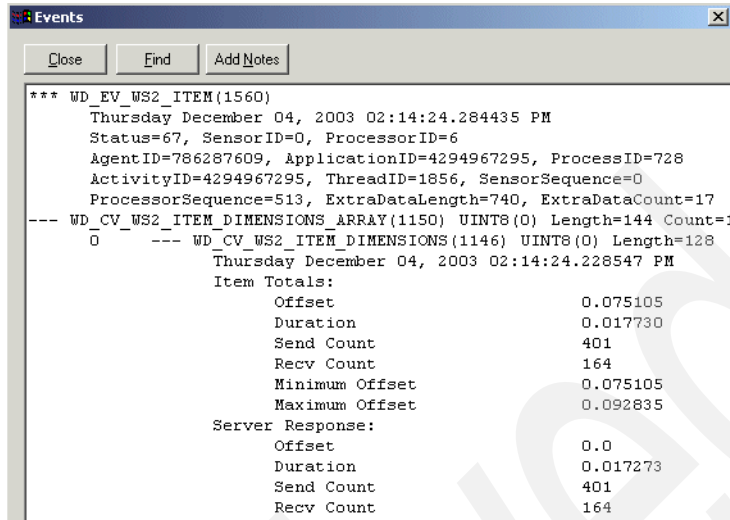


Figure 15-37 Page Detailer Events

In addition to the Chart view, it is also possible to view more details of all HTTP requests using the Details view. This can be seen by selecting the **Details** tab at the bottom of the window (see Figure 15-38).

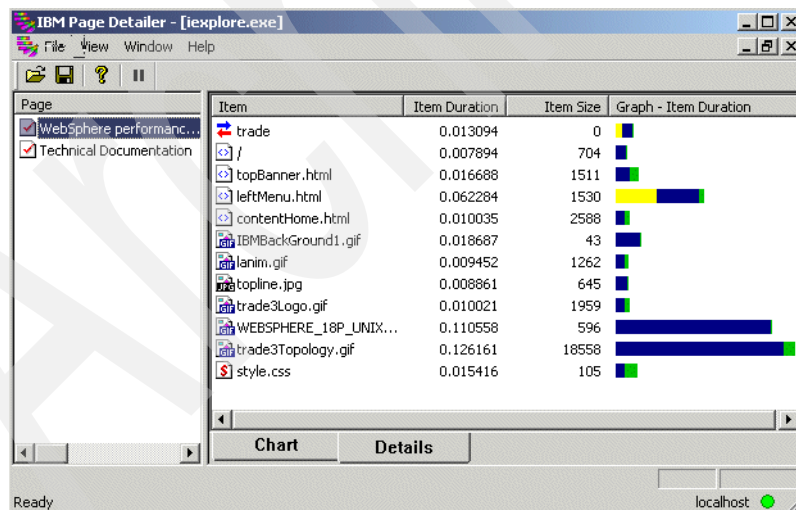


Figure 15-38 Page Detailer Details view

15.3.2 Important considerations

Some important considerations while taking measurements are as below:

- ▶ Impact of network delays

Many problems may not be evident when accessing a server on a local network, but may become apparent when accessing the site remotely, particularly when using a modem connection. On the other hand, you can minimize the effect of external network delays by directly connecting on the Server's LAN. This will allow you to isolate the performance impact of a change made to the Web page.

- ▶ Browser cache

Disabling browser cache helps in getting repeatable results. However you could also check the performance from a user's perspective by enabling the browser cache and comparing both results.

- ▶ Packet loss

Packet loss can happen and get corrected in the underlying TCP/IP layers. This is invisible to the Page Detailer. Packet loss manifests itself as inconsistent time measurements in Page Detailer. You can take a series of measurements at different times to factor it out.

15.3.3 Key factors

Some of the key factors that influence the time to load a Web page in a browser are:

- ▶ Page size.
- ▶ Number, complexity, and size of items embedded in the page.
- ▶ Number of servers that need to be accessed to retrieve all elements, and their location and network connectivity.
- ▶ Use of SSL (this introduces an extra overhead).

The Page Detailer will help you to identify when one of these problems is affecting some or all of your application. It will also help to identify problems such as broken links and server timeouts.

Some of the strategies that can be used to improve performance and resolve problems you have identified include:

- ▶ Minimize the number of embedded objects. Avoid the excessive use of images in particular. In cases where there is a standard header, footer, or side menu on every screen, consider the use of frames so that common elements do not have to be downloaded every time.

- ▶ The browser will typically retrieve multiple items in parallel, in the order in which they appear in the HTML page that it receives. Hence sequencing of the items so that downloads for larger objects are started early can reduce the total time required to display the page, and avoid the user having to wait for a long time for the last element(s) to be retrieved.
- ▶ Ensure that caching is being used effectively. Often the same images are used multiple times on the same page. If there are two references to the same image in close proximity to each other in the HTML source, the browser may encounter the second reference before the HTTP request that was initiated to download the first reference has been completed. In this case the browser may issue another request to retrieve the image again. This can be avoided by pre-loading frequently used images multiple times early, or by structuring the generated pages so that such URLs do not appear consecutively.
- ▶ Minimize the use of SSL where possible. For example, some content such as images may not need to be secured even though the application as a whole needs to be secure.
- ▶ Try to avoid switching the user to an alias server name during the page load. This will help the browser to reduce the look up time and possibly avoid a new connection.

15.3.4 Tips for using Page Detailer

Here are a few helpful tips to analyze the performance data shown by Page Detailer. A sample analysis, with comments, is shown in Figure 15-39 on page 892.

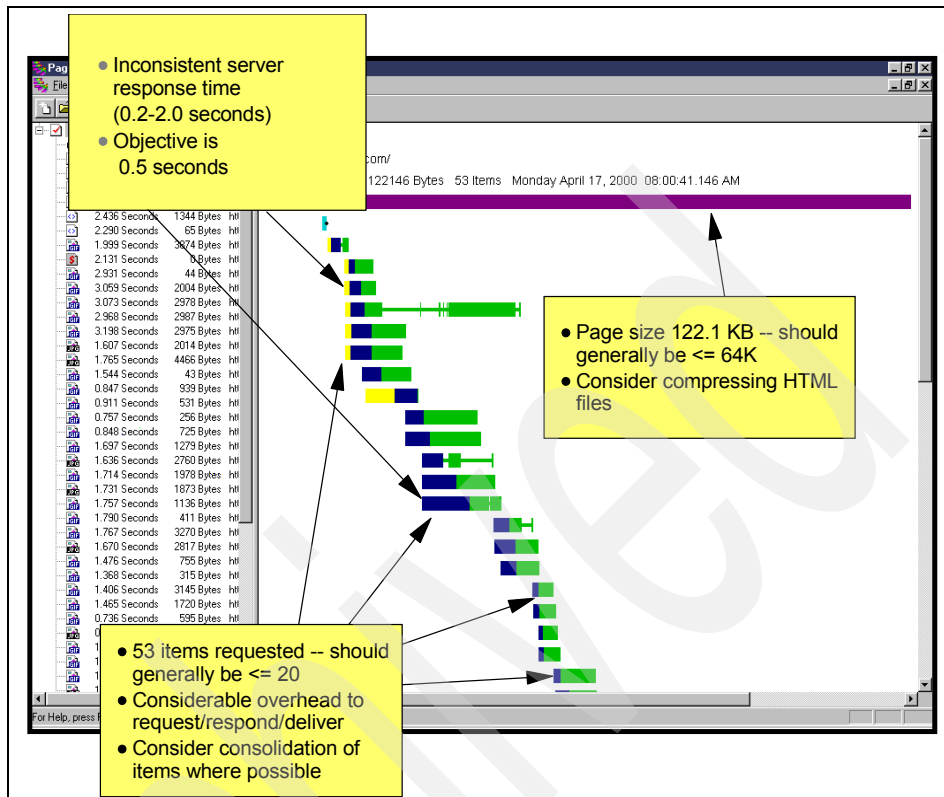


Figure 15-39 Page Detailer Sample Analysis

- ▶ A summary of above information and the meaning of each icon and color can be obtained by using the menu **View -> Legend**. This displays the window shown in Figure 15-40.

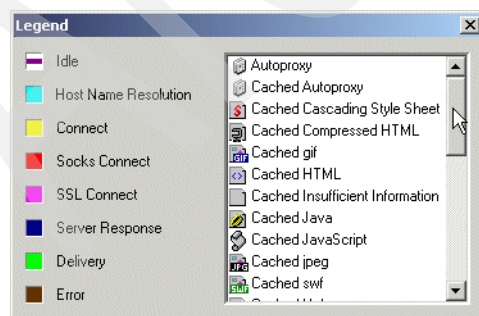


Figure 15-40 Page Detailer Legend

- ▶ Use a separate browser instance to collect data from related Web pages in one file. This allows for easy retrieval of performance data of related Web pages for comparison. Please note that you can save your work only in the Pro version of the Page Detailer.
- ▶ You can select the columns to display in the Details view from the context menu (obtained by right-clicking). You can also choose to display the column as a graph if it contains a numerical value.
- ▶ You can move the vertical bar and make room to add new columns on the left hand side of the window as shown in the Figure 15-41 and Figure 15-42 on page 894.

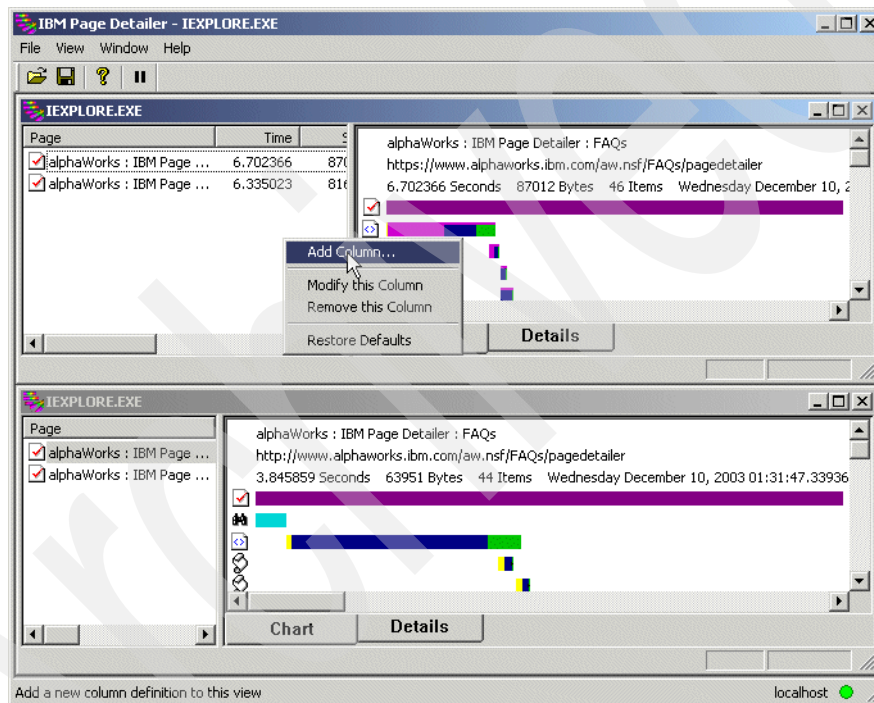


Figure 15-41 Moving the separator and adding a new column

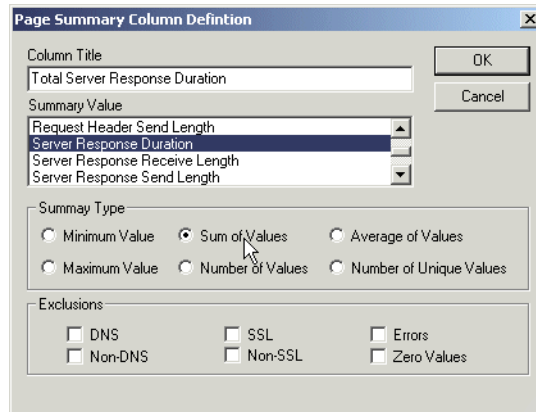


Figure 15-42 Column Definition

Tip: If it seems that Page Detailer is not collecting your browser interactions, try to navigate a little bit slower (there's a delay between the browser showing the page and Page Detailer capturing its content). The other point you must be aware of, is that Page Detailer installs itself configured for Microsoft Internet Explorer and several versions of Netscape browsers. If you want to use another browser, you need to check the documentation to find out, how to set it up.

15.3.5 Reference

- ▶ The article “Design for Performance: Analysis of Download Times for Page Elements Suggests Ways to Optimize” at:
<http://www.ibm.com/developerworks/websphere/library/techarticles/hvws/perform.html>
- ▶ The Page Detailer Web site at IBM alphaWorks:
<http://www.alphaworks.ibm.com/tech/pagedetailer>



Application development: best practices for application design, performance and scalability

This chapter discusses best practices for developing WebSphere Application Server-based applications.

16.1 Introduction

This chapter discusses best practices for designing and developing high-performance WebSphere Application Server based applications.

Before you start reading, you should be aware that the “best” in best practices is situational. There are certain situations where design decisions that are contrary to the “best” practices listed below are actually good. Just keep in mind that there are always several different factors that lead to a specific design decision, so being clear about the factors contributing to the decision is key.

We use the *Model View Controller (MVC)* paradigm as a basis for all design guidelines, because this common architectural concept is used in nearly all J2EE-based applications running on an application server. Figure 16-1 briefly describes the MVC-architecture: the Web-tier *controller* receives each incoming (HTTP) request and invokes the requested business logic operation in the application *model*. Based on the results of the operation and state of the model, the controller then selects the next view to display. Finally, the *view* renders the contents of the model. It accesses enterprise data through the model and specifies how that data should be presented. It is the view's responsibility to maintain consistency in its presentation when the model changes.

The main advantage is the clear separation between the different design concerns: data persistence, business logic, presentation and control. This means that changes to the presentation (view) of the data are limited to this layer; the other layers (business logic, control and data persistence) remain unaffected. The same applies to changes in all other layers. Another advantage is that this separation facilitates parallel application development.

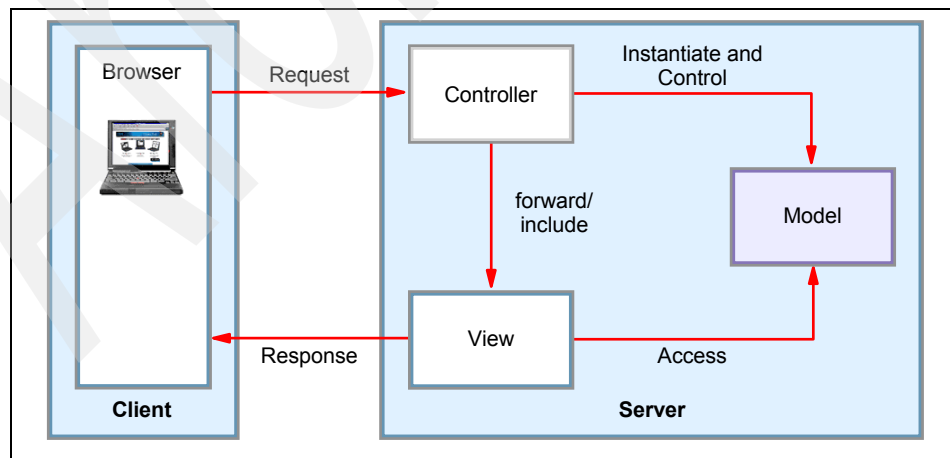


Figure 16-1 Model View Controller architecture

Nevertheless, even with a good architecture, it is still possible to have a bad design. Therefore, we outline several different design alternatives that are based on this model. For each specific layer, we focus on existing design patterns, rather than “reinventing the wheel,” but also cover new technologies that are available in IBM WebSphere Application Server V6. Throughout this chapter, we especially focus on performance-related aspects. In general, the best practices should help you in designing a scalable, high-performing application that can also be run on a highly available WebSphere cluster. In addition, we also cover some techniques and strategies that will assist you in optimizing the performance of your current application. However, they will not compensate for a poorly designed or architected application.

There can be a number of reasons for a poor application design, with limited scalability or several performance bottlenecks. These include a lack of focus on performance during requirement analysis and application design, a limited awareness of performance issues by the developers, or a desire to produce an elegant, extremely flexible or highly object-oriented solution. It is important that performance targets be set at the beginning of the project and communicated to all team members involved in development. The implications of these targets on development work should also be analyzed early in the development cycle and made known. Please note that there is often a trade-off between implementations that are elegant, flexible, easy to read and maintain, and those that offer higher performance.

In general, it is a good idea to verify as soon as possible that the performance expectations are met. In general, a load-performance test is a good way to find possible architecture and design defects. So it might make sense to develop and then load-test a prototype in a very early stage of the project, just to verify your design decisions. Always keep in mind that design changes in a later phase of the software life cycle are very, very expensive! Therefore, ongoing performance testing activities should be undertaken throughout the whole project life cycle to ensure that the performance requirements are met.

To optimize this specific testing of performance under simulated loads using a tool such as Rational Performance Tester 6.1, OpenSTA, or Apache JMeter is recommended. The tool can also provide insight as to where performance optimization work should be targeted.

Note: For more information about stress testing an application, refer to 17.1, “Testing the performance of an application” on page 940.

If your application is already in or near production and you notice performance problems, the following chapter also contains useful information about, for example, caching technologies or other performance optimizations IBM

WebSphere Application Server V6 offers to improve your application performance, without changing the overall application design. In these cases, profiling tools such as those included with IBM Rational Application Developer V6.0 and described in Chapter 15, “Development-side performance and analysis tools” on page 839, can also be extremely useful. These tools indicate which parts of the code are most frequently used, and also where the majority of the execution time is spent. Typically, the majority of the execution time will be spent in a minority of the code, as suggested by the 80/20 rule: 80% of the execution time will be spent on 20% of the code. In many cases, the ratio may be even higher, such as 90/10 or more. Although it is important to be aware of best practices when performing all development work, extra care should be taken to optimize the most frequently used code sections.

More information about server-side tools for analyzing usage patterns of components of an application such as servlets and EJBs can be found in Chapter 14, “Server-side performance and analysis tools” on page 769.

16.2 Presentation layer

The presentation layer is responsible for rendering the *view*, meaning the graphical representation of the user interface. In this section, we briefly discuss several different approaches of generating the response that describes the user interface, focusing on performance aspects.

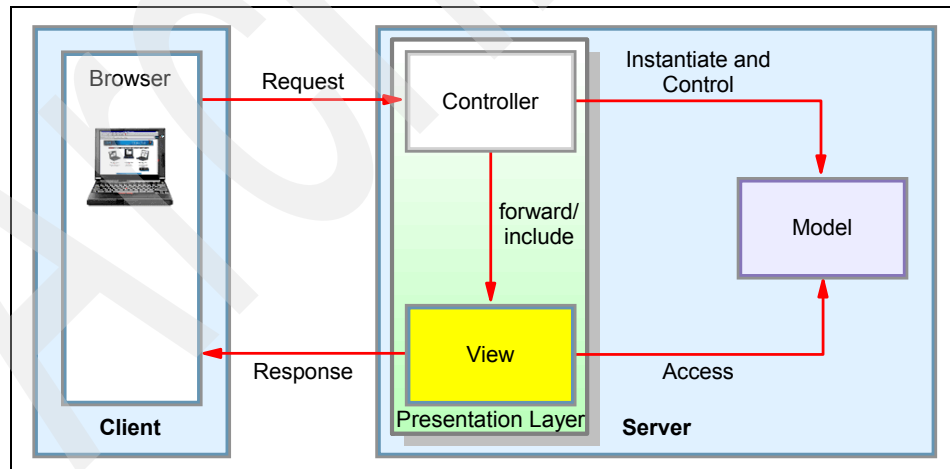


Figure 16-2 Presentation layer

16.2.1 JavaServer Pages

JavaServer Pages (JSP) technology is probably the best supported and best known presentation technology for Web applications available. JSPs were introduced to provide a clear separation between Java code of the application and the view, meaning the layout of the output pages. With the rollout of custom tag libraries, the JSTL, and the new JSP 2.0 features, it is becoming increasingly easy to build JSPs that do not require any Java code. There is also broad tooling support, including debugging support, for JSPs available in most of the development environments like WebSphere Studio, that makes it much easier for developers to work in a WYSIWYG way. There are several frameworks available, such as Struts, that use JavaServer Pages as their base for rendering their view.

JavaServer Pages usually offer the best performance of all technologies discussed in this part. Therefore, JavaServer Pages based frameworks should be considered first when building high-volume Web sites. Simple Web applications built on JSPs and servlets are faster than more complex frameworks like Struts or JavaServer Faces which will be covered later, but these have other significant advantages; for example, they simplify a good application design and are easier to maintain.

Since servlets and JavaServer Pages (JSPs) can include Java code, many of the issues discussed in other sections of this chapter are relevant to JSPs as well, especially the best practices for servlets, which are covered in the controller section of this chapter, 16.3, “Control” on page 907, and which are also applicable for JSPs. However, there are some particular issues that need to be considered when developing JSPs:

- Use composed JSPs to optimize caching and code re-use

JSPs that are composed of several other JSPs on one hand frequently use the `<jsp:include>` tag and therefore offend the best practice listed below, but on the other hand, the different components can easily be cached and re-used. So the use of caching can reduce the performance disadvantages of compositional JSPs, while facilitating the development of complex pages. WebSphere Application Server provides a functionality called *Dynamic caching service* to cache JSPs, thereby making it possible to have a master JSP which includes multiple JSP components, each of which can be cached using different cache criteria. For example, think of a complex portal page, which contains a window to view stock quotes, another to view weather information, and so on. The stock quote window can be cached for five minutes, the weather report window for ten minutes, and so on.

- Minimize the use of the `<jsp:include>` tag (or try to cache JSP components as mentioned in the previous list item about using composed JSPs)

Each included JSP is a separate servlet.

- Use `<jsp:usebean>` only to obtain existing objects

When a `<jsp:usebean>` tag is encountered and an existing Java bean object with the appropriate name does not exist, a new one is created. This is done by a call to `Beans.instantiate()`, which is an expensive operation because the JVM checks the file system for a serialized bean. Hence, it is recommended that the `<jsp:usebean>` tag only be used to obtain a reference to an existing object, rather than for creating a new object. When referencing a bean, you can use the following scopes:

- **page:** The bean is valid until the page sends the response to the user.
- **request:** The bean is valid from any JSP page processing the same request. You can use the request object to access the bean.
- **session:** You can use the bean from any JSP page in the same session.
- **application:** The bean is valid in any JSP in the same application.

Use the page scope whenever possible:

```
<jsp:useBean id="trainBean" scope="page"/>
```

- Do not create `HttpSessions` in JSPs by default

In accordance with the Java 2 Enterprise Edition (J2EE) specification, when executing a JSP, a session object is normally created implicitly if one does not already exist. However, if the session is not required, creation can be avoided by the use of the `<%@ page session="false" %>` directive.

- Do not use `SingleThreadModel`

Avoid the use of the `SingleThreadModel` for servlets, since it permits two threads to execute concurrently in the servlet's service method. The servlet container will handle `SingleThreaded` servlets either by synchronizing access to a single instance of the servlet, or by maintaining a pool of servlet instances and dispatching each new request to a free servlet. This reduced concurrency can significantly reduce the throughput, and consequently increase the response times experienced by users. If a servlet has shared variables that need to be protected, it is preferable to do so using synchronization of the relevant accesses. The `SingleThreadModel` is effectively equivalent to synchronizing the servlet's entire `service()` method.

- Use the `servlet.init()` method

The `javax.servlet.Servlet.init()` method can be used to perform expensive operations that need to be performed once only, rather than using the `doGet()` or `doPost()` methods of the servlet. By definition, the `init()` method is thread-safe. The results of operations in the `HttpServlet.init()` method can be cached safely in servlet instance variables, which become read-only in the servlet service method. A typical use for this would be to cache any JNDI lookups.

- Use the `HttpServlet destroy()` method

As you used the `init()` method for expensive operations and to cache some data as JNDI lookups, you should use the `destroy()` method to release these resources to avoid memory leaks. This method gives you an opportunity to clean up any resources that are being held (for example, memory, file handles, threads) and make sure that any persistent state is synchronized with the servlet's current state in memory.

For more information about JavaServer Pages, refer to the JavaServer Pages 2.0 Specification, found at:

<http://jcp.org/aboutJava/communityprocess/final/jsr152/index.html>

16.2.2 Struts

Struts is perhaps the most widely known framework for building Web applications, using the JSP approach as a foundation and enhancing it with a powerful custom tag library that implements the concept and the advantages of a Model-View-Controller based architecture. It is an open source framework that is part of the Jakarta project, which is sponsored by the Apache Software Foundation. Struts, like JavaServer Faces (see 16.2.3, “JavaServer Faces” on page 903), is an attempt to apply the Model-View-Controller (MVC) architectural pattern to Web application development. The Struts framework includes several pieces which make up the controller component of its applications, such as a central servlet (the `ActionServlet`) and developer-defined request handlers. Struts comes with a JSP UI tag library which supports the view component of its applications. The view of a Struts application may also make use of JSTL, XSLT, and Tiles, among other technologies. The application model is made up of Java beans, created by an application developer, which can make use of a variety of technologies, such as JDBC or EJBs. Struts introduces the concept of actions, which are used as request handlers when a special URI is used for form submission. These actions are responsible for processing form data (beans), possibly creating new form beans, and redirecting to a new JSP page. These actions act as a developer-defined intermediary between the Struts `ActionServlet` and the application model.

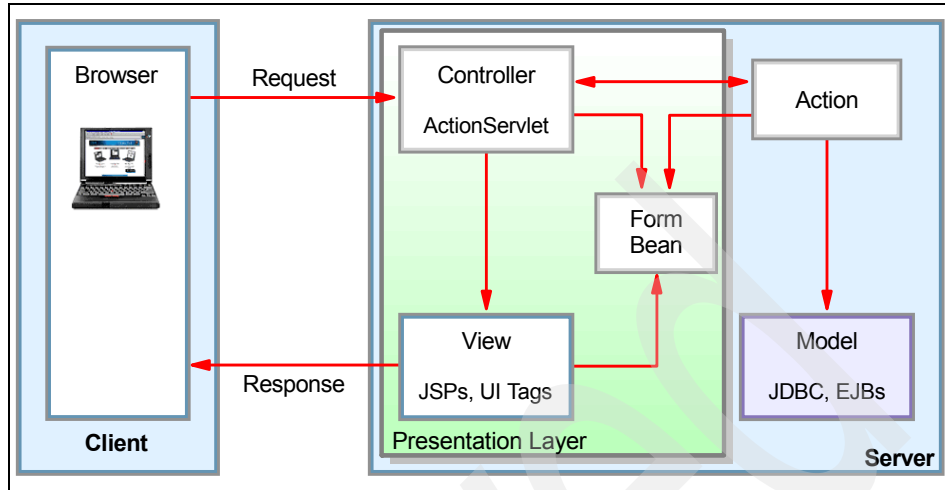


Figure 16-3 Struts architecture overview

In general, the Struts Framework has proven its performance and scalability. It is widely used and it also enforces the Model-View-Controller architecture, which is a big advantage compared to JSP-servlet based applications. Struts makes heavy use of JSP tag libraries, which results in a small performance overhead. This overhead can be easily reduced by using the dynamic caching framework of IBM WebSphere Application Server V6, which now includes support for the Struts framework, including tiles. For further information about this, see 16.2.5, “Caching for the presentation layer” on page 907 and 10.2.3, “Caching Struts and Tiles applications” on page 510.

For design and performance reasons, the following guidelines are recommended:

- ▶ Choose JavaServer Pages instead of XSLT for rendering the view
If you decide to use the Struts framework, you should try to use JavaServer Pages instead of XSLT to generate the user interface, whenever possible, for performance reasons.
- ▶ Do not use form beans to transfer data to the business logic tier
Although it will generate a small performance overhead to copy the data into a custom created data transfer object, you should not use the view-helper class (form bean) to pass the data to the business logic. The use of form beans for this purpose creates a dependency on the Struts framework that you do not want to force into the Business Process layer. Therefore, try not to use form beans for data transfer to provide a clean separation between the layers and to avoid data conversion at the business logic layer. You might also want to use reflection to easily copy the data between the two objects.

- Use servlet/controller best practices to implement action handlers
Actions are multi-threaded like servlets. They communicate with the model, invoke business logic, and return the model objects to the view. Finally, they perform tasks very much like controller servlets. Therefore, you need to follow all of the best practices associated with servlets (see 16.3, “Control” on page 907).

For more information about the Struts Framework, go to:

- “Struts, an open-source MVC implementation”
<http://www.ibm.com/developerworks/ibm/library/j-struts/>
- Struts home page
<http://struts.apache.org/>

16.2.3 JavaServer Faces

JavaServer Faces (JSF) technology is a server-side user interface component framework for Java technology-based Web applications, which is now generally available in WebSphere Application Server V6. One of the greatest advantages of JavaServer Faces technology, compared to standard JavaServer Pages, is that it offers a very clean separation between behavior and presentation.

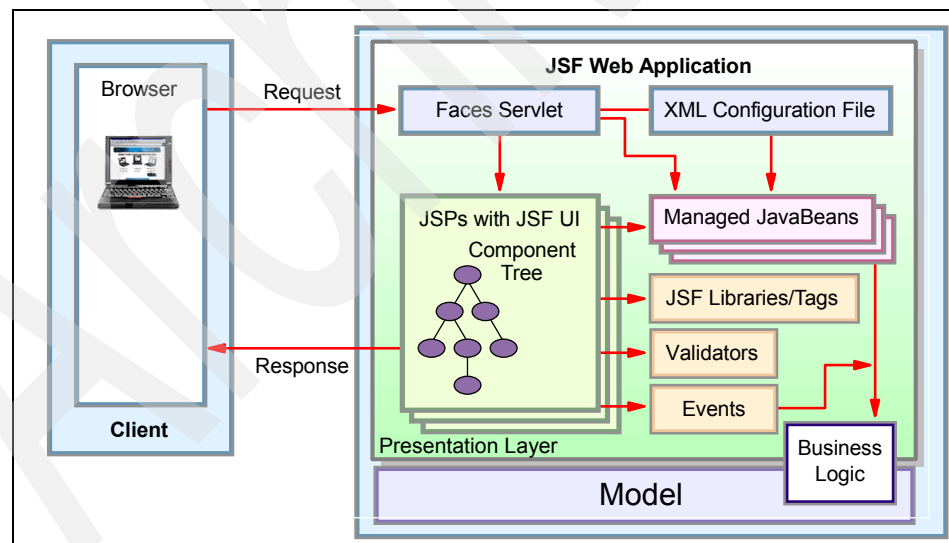


Figure 16-4 JavaServer Faces architecture overview

This means that the look (rendering) of a *User Interface (UI)* component and its feel (behavior) are now split up, which makes it possible to generate different

layouts (while implementing multiple renderers) for one and the same UI component. You can therefore easily add support for multiple different clients to a JavaServer Faces application. Another important goal of JavaServer Faces technology is to deliver, compared to, for example, JavaServer Pages, a richer and more responsive UI experience. JavaServer Faces components automatically provide their own event handling, support client- and server-sided validation and maintain their UI state on the server. Page navigation, data conversion and internationalization support are also included in the JavaServer Faces framework. This is done by adding a backing bean to each page of the application. A backing bean is a Java bean that defines all properties and methods from the associated UI components used on this page.

In general, the JSF Framework is very powerful and easy to use. One reason is that the JSF's reference implementation includes an extensive JSP tag library; there is also very good tooling support from several different products, such as IBM Rational Application Developer V6.0. Even though the reference implementation demonstrates JSF with JSP, it is important to realize that JSF is not exclusively tied to the JavaServer Pages technology.

Despite all the advantages of the new JavaServer Faces technology, there are also a few drawbacks:

- ▶ JavaServer Faces is a relatively new technology, so it has not proven its performance and stability in very large projects.
- ▶ First performance tests indicate that although the faces framework is more powerful, it is just a little bit slower than JavaServer Pages or Struts, because of its higher level of abstraction and the more complicated life cycle. When a client makes a request for a page containing JSF UI components, the JavaServer Faces implementation must perform several tasks, such as validating the data input of all components in the view and converting input data to types specified on the server side.

The performance and scalability of the JavaServer Faces Framework really depends on the implementation of the UI components you are using. If, for example, one UI component stores the state of thousands of table rows in its backing bean and therefore takes one minute to render the view, you cannot blame the JavaServer Faces architecture for this. So if you add custom UI components to your application, always check if they are optimized for performance and scale well.

- ▶ Control the amount of data that is stored in your backing beans.

JavaServer Faces implementations often provide very powerful UI components that make it easy for you to display data from your back end on the Web. Make sure that the amount of data that should be rendered does not get too much. For example, do not misuse a UI component to fetch a whole database table and store it in the backing bean, just to display some entries.

This might work well for a small number of users, but neither scales well nor satisfies performance expectations for a larger number of users.

For more information, please refer to the following sources:

- ▶ IBM Redbook *WebSphere Studio 5.1.2 JavaServer Faces and Service Data Objects*, SG24-6361.
- ▶ *Integrating Struts, Tiles, and JavaServer Faces* document at:
<http://www.ibm.com/developerworks/java/library/j-integrate/>

16.2.4 XML/XSLT processing

With the help of the *Extensible Stylesheet Language Transformation (XSLT)*, which is a part of the XSL standard, you can easily transform any XML-based document into another. In the presentation layer, this technology is usually used to generate an XML-based document that is recognized by a browser, like HTML, XHTML or other XML-based documents that can be understood by other client devices such as handheld devices. To do this, XSLT first parses the source XML document to determine which parts have to be transformed. The parts that should be transferred, and the manner of transformation, are pre-defined in one or more templates. So if a matching part is found, XSLT will transform this part of the source document into the resulting XML document. All other parts that do not match any of the templates remain unmodified and are simply copied into the result.

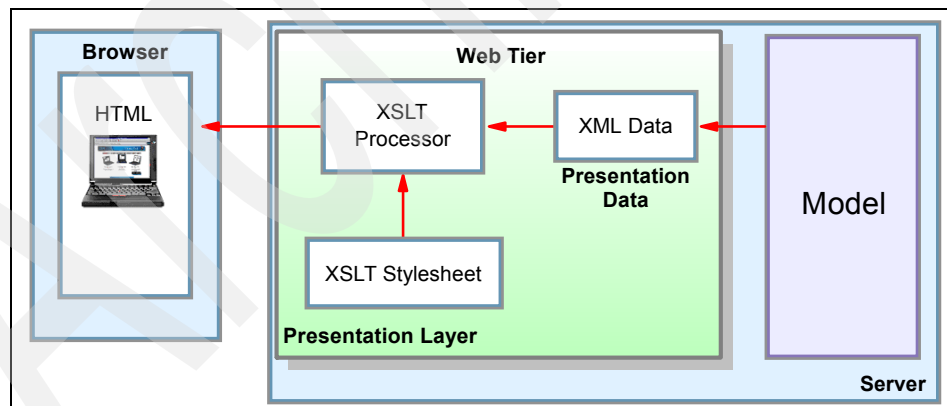


Figure 16-5 Server-sided XSLT processing

In general, the XSLT transformation can take place either on the application server or in a browser/client that supports XSLT. We only cover the server-side XSLT processing, because XSLT processing on the client is not possible in most cases due to security reasons, network traffic, or limited browser support. A valid

compromise could be to implement a servlet that checks the client for an XML-enabled browser and returns the XML directly to the client to get client-side XSLT. Server-side XSLT is only done when the client is not XML-enabled.

Server-sided XSLT processing

The use of XML/XSLT processing on the application server to generate the view of your application implies a lot of processing overhead and, from a performance point of view, it is therefore only recommended in cases where you really have multiple presentation output types that must be supported. Performance tests done at IBM comparing the relative speed of XSL and JSP show that in most cases, a JSP will be several times faster at producing the same HTML output as an equivalent XSL transform, even when compiled XSL is used. While this is often not an issue, in performance-critical situations, it can create problems.

This does not mean that you should never use XSL, but it might not be the recommended way to generate the view for high volume Web sites. However, there are certain cases, especially in the pervasive computing sector, where XSLT might be the best/easiest solution for rendering the views. Here the power and the abilities of using XSL to support multiple mobile devices countervail the processing overhead by far. But this kind of requirement is most often the exception rather than the rule. If you are using XSLT just for producing the HTML rendering for each page, then this is overkill and will cause more problems for your developers than it will solve.

If you decide to use XSLT processing on the server, you should use the following best practices to minimize the performance impact:

- ▶ Use XSLTC, the compiled version of XSLT, whenever possible
XSLTC directly compiles the stylesheet into a Java class and is therefore the much faster alternative. It is about three times faster than the XSLT interpreter.
- ▶ Keep XSL stylesheets as simple as possible; use XML Data Transfer Objects
It is a common best practice to generate a value object to transfer the data between the business logic layer and the presentation layer. The same should be done using XSLT. The business logic should return a data transfer object (or SDO) that contains the XML representation of the data needed in the view. There is no need to have any complicated logic in the XSL stylesheet to generate or collect the XML data that is needed for the view. The generation of an XML data transfer object really simplifies the stylesheet processing and greatly improves performance.

16.2.5 Caching for the presentation layer

Server-side caching techniques have long been used to improve Internet performance of Web applications. In general, caching improves response time and reduces system load. Since the introduction of dynamic caching, WebSphere Application Server is also able to cache dynamic content that changes from time to time. Although dynamic caching can be added later to nearly any Web-based application, because the programming model is unaffected, it requires a proactive and effective invalidation mechanism to ensure the freshness of the content. To maximize the performance improvement, it might also make sense to decompose the pages into several small cachable units (see JSP best practices in 16.2.1, “JavaServer Pages” on page 899). Therefore, you should plan to integrate the IBM dynamic caching technology as soon as possible in your development life cycle. Although JSP/servlet caching is an IBM extension, and is not part of the current J2EE specification, the application changes for invalidating cache entries are marginal. Therefore, the application is still portable and compliant with the J2EE specification, but still benefits from the performance optimizations provided by the dynamic cache. The cache policy itself is specified declaratively and configuration is through XML deployment descriptors.

IBM WebSphere Application Server V6 now provides support for the following presentation technologies:

- ▶ JavaServer Pages/servlets
- ▶ Struts and Tiles

For further information about Dynamic Caching, look at Chapter 10, “Dynamic caching” on page 501.

16.3 Control

Whatever the presentation technology, requests for domain state and behavior will be done through a Controller object defined for the particular presentation requirements.

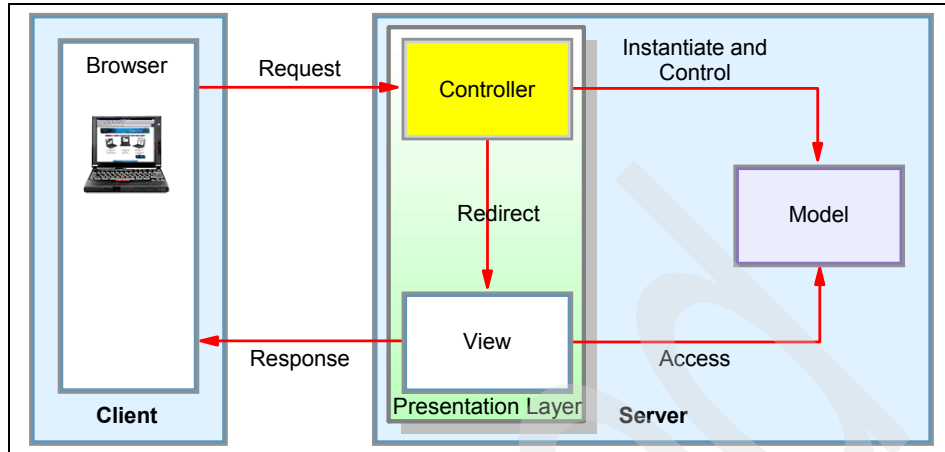


Figure 16-6 Control

For Web-based application front ends, servlets are usually used to implement this work. If you are using a framework, like Struts or JavaServer Faces, controller objects based on Java servlets are already defined to handle the incoming client requests. These frameworks provide a clean separation between the presentation and the control objects so if you decide not to use one, you must take care of this yourself.

For server-based applications that support “fat clients,” meaning clients that have to be installed on every client machine, the implementation of the controller object depends on the communication protocol that is used. If you are using HTTP or HTTPS, for example SOAP/HTTP or SOAP/HTTPS, to communicate with the application server, you might also want to use Java servlets to control incoming requests. If RMI/IIOP or JMS is your preferred protocol to send requests to the application server, Enterprise JavaBeans, more precisely EJB session beans, are the best fit to handle incoming RMI/IIOP traffic or EJB message-driven beans for JMS.

Irrespective of the technology that is used to control the incoming requests, the following guidelines should be observed:

- ▶ Keep controller objects as simple as possible

The controller should neither contain any business logic, nor generate any kind of presentation. Its purpose is just mediation between the presentation and the business logic layer and conversion from one interface to another.
- ▶ Abstract Parent Class for all servlets

Consider creating an abstract parent class, using the template inheritance pattern, for all your controller servlets (and Struts action handlers) to realize a

common behavior. This abstract class is a good place to put standard code that you want executed in *all* servlets, such as tracing, logging or additional security.

- Avoid the "killer" servlet

Use multiple servlets instead of one massive one. This avoids undue routing logic and also allows you to configure J2EE security for individual servlets. This guideline also simplifies maintenance and team development process.

16.3.1 Maintaining state: stateful session beans versus HTTP session

Because the protocols used for communication between the client and the application server are usually based upon a request/response model and therefore stateless, the application developer has to take care of the state himself. For example, if a request is submitted from the browser to the application server, the server will just receive the request, process it and send a response back. After this transaction is complete, there is no way for the protocol to hold state information about the transaction itself. Therefore, this state information has to be stored either on the client or on the server. If you decide to store this information on the application server, another problem arises, because usually all resources are shared and pooled. Therefore, you never know which servlet or session bean receives your next request to store the state there. The following section compares the two most common approaches for storing session data to maintain state on the server: stateful session beans and the HTTP session.

In general, HTTP sessions might better fit your needs if you are building systems that only require a Web front end.

Stateful session beans usually simplify application development if you have to support an EJB client or you need a stateful object that is transactional aware.

HTTP session

In a Web application, state information relating to each client is typically stored in an HTTP session, which is identified by some unique identifier that is associated with an HTTP cookie. In an environment with a single application server, session information can be stored in-memory by IBM WebSphere Application Server V6.

However, it is more common to use a clustered environment with multiple application servers to provide scalability and improve fault tolerance. In this scenario, session information needs to be made available for multiple or even all cluster members. In WebSphere Application Server V4.x and earlier, this was achieved using a session persistence database that was available to all clones in a server group. In addition to this, a new mechanism for memory-to-memory replication has been introduced in IBM WebSphere Application Server V5.0.

Please refer to 1.4, “Managing session state among servers” on page 24 and “Session persistence considerations” on page 78 for more information about session management, database session persistence and memory-to-memory replication.

The following guidelines are important to ensure performance and scalability while using HTTP sessions:

- Keep HTTP sessions small

HTTP sessions should only be used to store information about application state; it is not a data cache! You should always try to minimize the amount of data stored in the session. Since the session must be shared, it must be serialized, which also involves serializing all objects that are reachable from the session. Serialization in Java is an expensive operation. If persistent sessions are used, the serialized session data must be stored in the database (usually as a BLOB), which introduces further overhead. These operations of storing/recovering state, which are based on object serialization, can be improved by using Java externalization. Externalization may be up to 40% faster than serialization. Nevertheless, in order to reduce the amount of data stored in the session, avoid storing large, complex object graphs in it. Sometimes, it may be beneficial to store objects in the session, although they can be recreated or retrieved to avoid the overhead of doing so. In these cases, consideration should be given to making these attributes transient. If this is done, you have to ensure that the application code will handle the transient attributes having null values. Alternatively, the `readObject()` method of the object could be overwritten to recreate the transient data when the object is deserialized. Again, Java externalization can be used to eliminate introspection and improve performance.

- Always invalidate unused HTTP sessions

The session object can be garbage collected after it has been invalidated. This can be done programmatically or after a predefined time-out period during which the session was not accessed. To allow the memory used by the session to be reclaimed as early as possible, it is best to explicitly invalidate the session when finished with it rather than waiting for the time-out. This may require the introduction of logout functionality into the application, and training for the users to make use of this functionality rather than simply closing the browser.

- Do not cache references to HTTP sessions

References to the session should always be obtained from the current servlet context as required; they should not be cached by the application. This ensures that the session objects can be reclaimed when the session is invalidated. Another reason for doing this is that a session reference is not guaranteed to be valid outside of the context of a specific server interaction.

- Take care when using HTML frames

Special care must be taken when using HTML frames when each frame is displaying a JSP belonging to a different Web application on the same server. In this case, a session should only be created and accessed by one of the pages. Otherwise, although a session will be created for each page, the same cookie will be used to identify the session. This means that the cookie for each newly created session overwrites the previous cookie, and only one of the sessions will be accessible. The remaining sessions will be created but will be inaccessible and thus will consume memory until the time-out interval is reached. If the Web application was split into multiple applications in order to improve scalability, consider combining all of the Web applications into a single one, and using clustering to achieve the required scalability.

There are alternatives to the use of sessions, besides the stateful session beans, that may be appropriate in some situations:

- Hidden form fields or cookies

In some cases, use of the session can be avoided by using hidden form fields or cookies to store data. Note that there is a 4 KB limit on the total size of all cookies for a particular site. Also, be aware that the use of hidden fields increases the page size and the data can be seen by the user when viewing the HTML source.

- Defer persistence of session data to the business logic

Data can also be persisted into a database by the business logic. By using native data types instead of serialized BLOBs, it is often possible to achieve better performance. It is also possible to read and write only the data that has changed, rather than the entire data set as is normally the case with BLOBs. The application must remove data when it is no longer required (after a time-out period). This can be implemented by placing an object that implements the `HttpSessionBindingListener` interface into the session, and placing the clean up code in the `valueUnbound()` method.

Stateful session beans

Stateful session beans are, unlike stateless session beans, not shared among multiple clients. In addition, stateful beans are not pooled. A stateful session bean is dedicated to one client for the life cycle of the bean instance, the client just needs to store the reference to it. This allows you to store the state in instance variables of stateful session beans for the life of the instance.

The following guidelines apply to stateful session beans:

- Remove stateful session beans when finished

Instances of stateful session beans have affinity to specific clients. They remain in the container until they are explicitly removed by the client, or

removed by the container when they time-out. Meanwhile, the container might need to passivate inactive stateful session beans to disk. This is overhead for the container and constitutes a performance hit to the application. If the passivated session bean is subsequently required by the application, the container activates it by restoring it from disk. By explicitly removing stateful session beans when finished with them, applications decrease the need for passivation and minimize container overhead. To remove a stateful session bean, simply use:

```
mySessionBean.remove();
```

- Keep the size of stateful session beans as small as possible

For stateful session beans, nearly the same size restrictions than for HTTP sessions apply. They should only be used to store the application state. If stateful session beans are used in a clustered environment, the replication service responsible for maintaining state across the cluster is based on the same technology as memory-to-memory replication of persistent HTTP sessions, therefore the same performance/size impacts are encountered.

16.4 Business logic layer

The previous sections covered different technologies and approaches to render the user interfaces and control incoming requests. The following section focuses on the most important part, the business logic.

The business logic layer is the core of the application, it implements the business process model that describes how work is performed. It is usually the part that needs most investment during the analysis and design project phases, because it contains lots of knowledge about how a business operates. This layer is responsible for processing the incoming data and generating the desired results. In this tier are the business objects and all the business rules necessary to perform the client operations. The business logic can be implemented using a lot of different technologies, such as Enterprise JavaBeans, Service Data Objects or JavaBeans. In addition to these technologies, this tier could be implemented using Web services, or even invoking a third-party Web service.

While designing this part of the application, you should focus on extensibility and providing a flexible and not too fine-grained interface, that can be reused by other applications. As a design guideline for the business logic interface, you should have a closer look at the service-oriented architecture (SOA). SOA is a flexible architecture that unifies business processes by structuring large applications into building blocks, or small modular functional units or services. Such a framework isolates each service and exposes only the necessary declared interfaces to other services. The SOA model isolates aspects of an application so that, as technology changes, services (components) can be

updated independently, limiting the impact of changes and updates to a manageable scope. Managing change is an important benefit of leveraging component architectures and models. If not managed well, change can result in the degradation of a modern Web application into unwanted complexity. A comprehensive design pattern can bring much needed structure to Web application development.

Even if you do not see the need for integrating your business logic into other applications today, this may change in the future. The service oriented approach will simplify the integration and implementation of a Web-services interface. The idea of separating your business logic into different services will also help you to define the granularity of the business logic interface. For example the methods `getLongValue()` and `getIntValue()` do not make sense for a business service, whereas `transferMoney(fromAccount, toAccount, lAmount)` does. The granularity of the interfaces also has a big impact on the overall performance of the solution, if they are too fine-grained, the communication and transactional overhead increases dramatically.

Note: If you want to learn more about SOA, go to

<http://www.ibm.com/developerworks/webservices/newto/index.html>

16.4.1 How to implement the interface to the business logic

As already mentioned, the implementation of the interface to the business logic greatly affects performance, scalability, extensibility, and maintainability of the whole application. Therefore, the design and the technologies used for this interface should be carefully evaluated.

We discuss the different approaches and technologies to implement this interface.

EJB session beans and message-driven beans

Enterprise JavaBeans (EJB), more precisely Session Beans and message-driven beans, allow developers to focus on writing the business logic necessary to support their applications without having to deal with the intricacies of the underlying middleware. The EJB container provides crucial services such as transactions, security, naming and lays the foundations for scalability.

The following possibilities, based on Enterprise JavaBeans, exist to access the business logic layer for both a Web based application or a pure Java client application or applet.

- ▶ Facade (see “Facade” on page 915)
 - Session facade

- Message facade
- EJB Command Framework (see “EJB Command Framework” on page 917)

Before we cover each of these concepts in detail, here are some general guidelines for using Enterprise JavaBeans:

Enterprise JavaBean guidelines:

- Cache EJB references

Since EJBs are able to be accessed remotely, obtaining a reference to an EJB involves a lookup process. This can take a relatively long time, and hence caching the references obtained can improve performance on subsequent lookup operations. Obtaining an EJB reference typically involves the following steps:

- a. Obtain an InitialContext instance.
- b. Obtain a reference to the home interface of a particular EJB by looking up the EJB via the initial context object and performing a narrow operation.
- c. Obtain a particular EJB instance by executing the appropriate finder or create method from the EJB home instance.

The calls to obtain an InitialContext instance and lookup of an EJB Home instance (steps a and b) are expensive, and performance can be improved by caching these objects. References to EJB homes may become stale, and hence it is best to wrap accesses to EJB home methods with code that refreshes stale references.

- Reduce remote method calls

Since EJBs are intended to be accessible remotely, calls to EJB methods are implemented as remote calls even if the EJB exists in a container that shares the same JVM as the Web container, which introduces some overhead. In some cases, the overhead can be reduced by implementing one of the approaches outlined below.

One of the key mechanisms to reduce EJB overhead is to minimize the number of remote calls. In general, a servlet should make a single EJB method call to a session bean to perform an operation. If required, this EJB method can call other methods as required to perform more complex operations. All access to entity bean EJBs should be performed through stateless session beans. Session beans should be used to implement the business logic, while entity beans should be used for data storage and retrieval.

IBM WebSphere Application Server V6 is compliant with the J2EE 1.4 specification, which mandates support for EJB 2.1. The concept of a local interface has been introduced for EJB 2.0. If local interfaces are used, the overhead of serializing the parameters and transmitting them via RMI/IIOP

(Remote Method Invocation/Internet Inter-ORB Protocol) is avoided. Parameters can be passed by reference instead of value, avoiding the need to copy them. In EJB 2.0, an EJB can have a remote interface, local interface or both, although typically a particular bean will have either a remote or a local interface. Since entity bean EJBs are normally accessed from session beans, in most cases they will only need local interfaces.

- ▶ Reduce the number of transactions

In addition to the overhead associated with remote method calls, transaction management provided by the EJB container can also introduce overhead. Accessing entity beans from session beans can limit the impact of this by reducing the number of transactions. In the deployment descriptor for EJBs, the transaction type can be specified. This can be one of five values: NotSupported, Supports, Required, Requires New and Mandatory. Set the transaction type to NotSupported if no transaction is required.

- ▶ Use `setEntityContext()` method to cache data

The `setEntityContext()` method is only called once, during the creation of each bean. So you can use this method to cache any data for the life of the bean. Use the corresponding `unsetEntityContext()` method to release any resource that you are holding in the bean, otherwise you will have this resource held forever.

For more information about this topic please refer to “Entity beans” on page 924.

16.4.2 Facade

The intent of facades is to provide a simple and unified interface to the otherwise complex business model that lies behind it. By doing so, the dependencies between the model classes and its clients is reduced. Less dependencies means more freedom to adapt to new requirements. In conjunction with this pattern, the Business Delegate pattern and the (Data) Transfer Object pattern are two other design patterns that are usually used within the business layer.

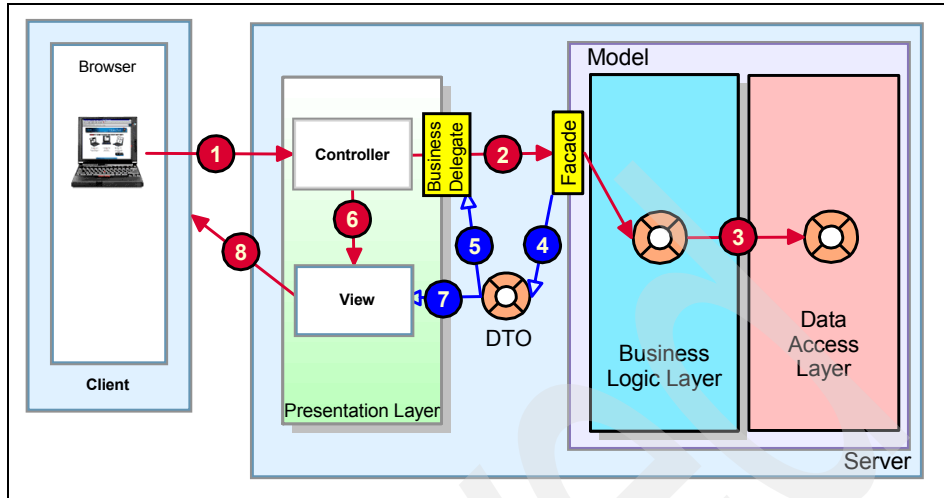


Figure 16-7 Overview of Facade including business delegate and Data Transfer Object

► Business delegate

The Business delegate represents the interface between the presentation layer and the business layer. It decouples the presentation logic from the EJB tier by providing a proxy to the EJB tier services. The delegate takes care of lower level details, such as looking up remote objects and handling remote exceptions.

► Data transfer object

Data transfer object (DTO) design pattern (refer to *EJB Design Patterns*, by Floyd Marinescu), also known as value object and transfer object. The idea is to limit inter-layer data sharing to serializable JavaBeans, thus avoiding remote references. This DTO can be created by the session facade or by a builder object (according to the builder design pattern) on its behalf, in case the building process is too complicated or needs validation steps.

We briefly describe the two different kinds of implementing a facade as stateless session beans or message-driven beans in the following section. In most cases you will want to use both, because both of them have their own unique advantages particularly with regards to performance and scalability.

The reasons for using a facade are the following:

- Simple, flexible and unified interface to the business logic
- Reduce unnecessary network and transactional overhead
- Reduce unwanted dependencies and coupling between the layers

Session facade

If you are using the session facade, the business layer provides the services to clients via stateless session beans. Each stateless session bean should at least have a remote interface, which is needed to enable load-balancing and failover at the business logic layer. Stateless session beans are transactional aware, scale well and are pooled in the EJB container.

The session facade is a synchronous way of accessing the business logic and it should be used whenever immediate feedback is required. This kind of feedback is in general required for read operations. For writing or updating the business model, it might make sense to have a look at the messaging facade.

Message facade

If you have a detailed look at your business logic, there may be some transactions that can be run asynchronously, for example a rental car reservation, where you get a confirmation by e-mail. In these cases, where an asynchronous call is sufficient, a message facade can improve the performance and scalability of your application a lot. The message facade is implemented with message-driven beans that also offer full transaction support. The message-driven bean is not accessible through a remote or local interface. The only way for an EJB client to communicate with a message-driven bean is by sending a JMS message. The container delegates the message to a suitable message-driven bean instance to handle the invocation. So try to use a message facade for write/update operations where you do not need immediate feedback.

16.4.3 EJB Command Framework

The EJB Command Framework also implements a kind of facade to provide a separation between the different layers. In contrast to the session facade, where each of the session beans or rather the business delegate interfaces need to be defined the same way on the different layers, the EJB Command Framework just provides a unique interface for all business logic processes. The business process that should be called and its corresponding data is encapsulated in a Java bean, called *command bean*. This command bean is the only parameter that is needed by the EJB session or even message-driven bean, that implements the facade to the business logic. Therefore, the interface to the business layer always remains the same, even if new business logic is added. There is no need to copy the regenerated EJB stubs or new remote interfaces from the new session facade to the other layers. You just have to add and implement the new command beans.

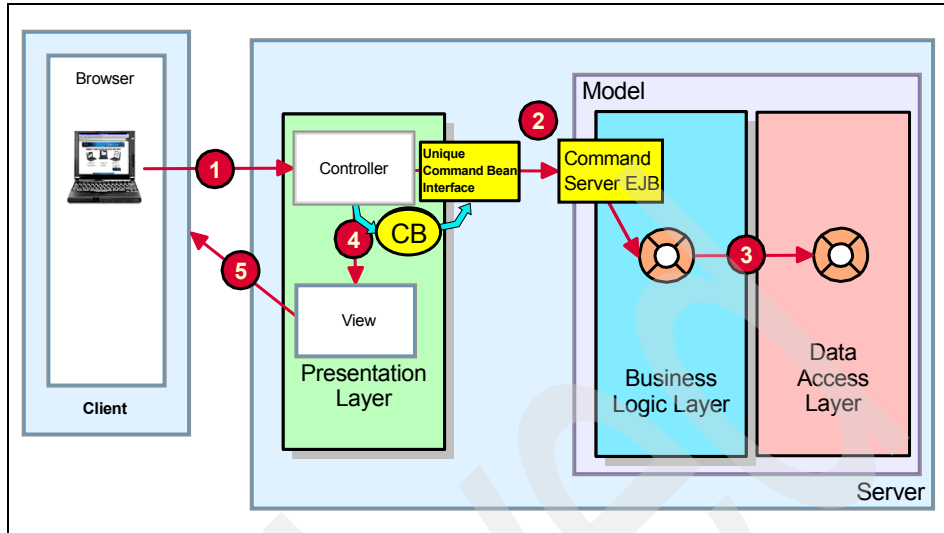


Figure 16-8 EJB Command Framework architectural overview

At first sight, the EJB Command Framework does not seem to have any disadvantages - it provides a very flexible interface for a loose coupling between the layers that even simplifies parallel development in large teams, it can be accessed synchronously or asynchronously and it also provides good scalability.

But there are also a few drawbacks. First, it is difficult to define the right granularity for the command beans. If there are too many of them, command bean management may get hard and the session bean that has to process the incoming beans gets more and more complex. In addition you also may want to add different methods to the session bean, for example:

```
executeWithTransaction(CommandBean bean)
```

or

```
executeWithoutTransaction(CommandBean bean)
```

to change the transactional EJB settings for the commands. In addition, it is difficult to implement a good and meaningful error handling for all kinds of command beans.

Note: WebSphere Application Server ships a complete command framework, which formalizes a solution to the above problem. The command facility is implemented in the `com.ibm.websphere.command` Java package. To find more information about the Command framework go to:

http://www.ibm.com/developerworks/websphere/registered/tutorials/0306_mcguinnes/mcguinnes.html

16.4.4 Caching business logic

WebSphere Application Server is able to cache the results of business logic calls. This is handled by the WebSphere command cache, which can cache the return parameters of any Java method call including remote EJB calls. The cache therefore significantly improves performance by reducing the number of remote or local calls to the business logic. Just as the servlet/JSP caching feature of the Dynamic Caching engine in WebSphere allows an HTML page to remain fresh for a period of time, or until it is explicitly marked dirty by an API call, a cached result from the business logic can also be held for a period of time or until it is explicitly invalidated. However, it is important that you use Dynamic Caching only when appropriate. It may consume more resources than it saves if you cache the wrong business logic calls, for example, calls whose parameters are rarely the same or that are not called again before the invalidation time fires. In general, you should try to identify methods that could be cached as early as possible to implement a appropriate caching strategy.

A good example for a cacheable business logic call is a `getLatestNews(topic t)` call that returns the latest news for a specified topic. The news are possibly stored in a database and updated every 15 or 30 minutes, but called frequently during this time!

16.5 Data access layer

The purpose of the data access layer is to provide a flexible and portable data programming model that separates the data access, which usually depends on some kind of Enterprise Information System (EIS), from the data itself. In addition, the data access layer decouples the application code from data access code, to enable business logic reuse and simplify application maintenance.

There are different technologies available that simplify access to persistent storage, such as Enterprise JavaBeans - Entity beans, Java Data Objects, Service Data Objects, or the native JDBC technology that also provides cross-DBMS connectivity to a wide range of relational databases. We explain the

different persistence frameworks and technologies and look at their strengths and weaknesses in the following sections.

16.5.1 Service Data Objects

Service Data Objects (SDO) is a data programming architecture that - in contrast to all other technologies that are currently available - unifies data programming across data source types. Using SDOs, application programmers can uniformly access and manipulate data from heterogeneous data sources, including relational databases, XML data sources, Web services, and Enterprise Information Systems.

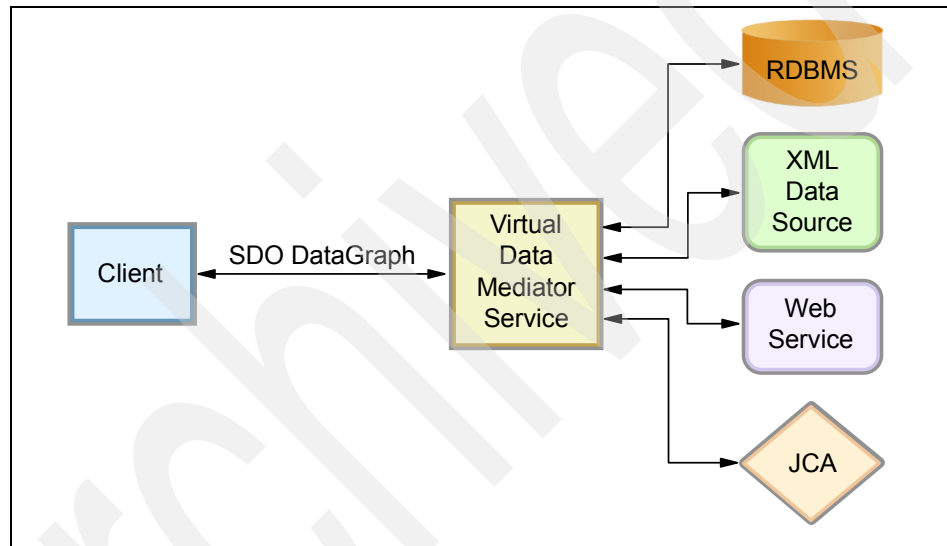


Figure 16-9 Flexibility of Service Data Objects

To realize this simple and unified way to handle data, Service Data Objects add a new abstraction layer that is placed on top of existing data access frameworks like EJBs, JDO, or direct JDBC data access. Therefore, Service Data Objects do not replace existing frameworks, instead, they use them as data mediators under the cover. In fact, Service Data Objects are becoming a standard way to implement the Business Delegate and Data Transfer Object patterns.

SDO Architecture

The SDO architecture consists of the following three major components:

- ▶ Data object
- ▶ Data graph
- ▶ Data mediator

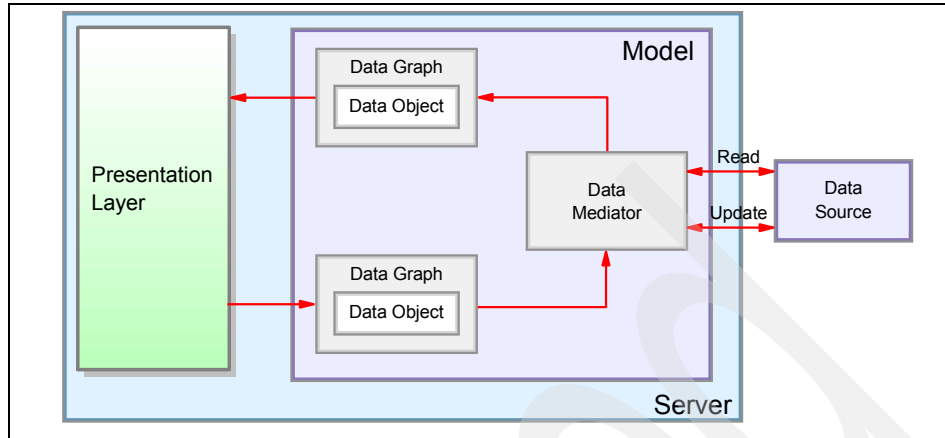


Figure 16-10 SDO architecture

► Data object

The data object is designed to be an easy way for a Java programmer to access, traverse, and update structured data. Data objects have a rich variety of strongly and loosely-typed interfaces for querying and updating properties. The implementation of the data object also handles data conversions if needed. Data objects store the data using a disconnected, optimistic model, meaning the data is available locally without an active connection to the EIS. Therefore, the data object can be easily used to transfer data between the different application layers. This enables a simple programming model without sacrificing the dynamic model required by tools and frameworks. A data object may also be a composite of other data objects.

► Data graph

SDO is based on the concept of disconnected data graphs. A data graph is a collection of tree-structured or graph-structured data objects. Under the disconnected data graphs architecture, a client retrieves a data graph from a data source, mutates the data graph, then applies the data graph changes to the data source. The data graph also contains some metadata about the data object including change summary and metadata information. The metadata API allows applications, tools, and frameworks to introspect the data model for a data graph, enabling applications to handle data from heterogeneous data sources in a uniform way.

► Data mediator

The task of connecting applications to data sources is performed by a data mediator. Client applications query a data mediator and get a data graph in response. Client applications send an updated data graph to a data mediator to have the updates applied to the original data source. This architecture

allows applications to deal principally with data graphs and data objects, providing a layer of abstraction between the business data and the data source.

Important: Update processing is not dependent on how the DataGraph was originally retrieved. In other words it is possible to retrieve a DataGraph directly from the data source but to have the deferred updates applied through an EJB or vice versa.

Regardless of which update approach you use, an optimistic concurrency control algorithm is used. Fields designated as consistency fields are read during update to insure that the current value is still equal to the old value of the field in the DataObject.

This means that mediators are components that provide access to a specific data source type. For example, a Siebel mediator knows how to mediate between changes made to an SDO and the necessary calls to the Siebel API to persist changes to the underlying Siebel records.

In IBM WebSphere Application Server V6 the following Data Mediator Services are currently available:

- Enterprise JavaBeans Data Mediator Service

The Enterprise JavaBeans (EJB) Data Mediator Service (DMS) is the Service Data Objects (SDO) Java interface that, given a request in the form of EJB queries, returns data as a DataGraph containing DataObjects of various types. This differs from a normal EJB finder or `ejbSelect` method, which also takes an EJB query but returns a collection of EJB objects (all of the same type) or a collection of container managed persistence (CMP) values.

- Java DataBase Connectivity Mediator Service

The Java Database Connectivity (JDBC) DMS is the SDO component that connects to any database that supports JDBC connectivity. It provides the mechanism to move data between a DataGraph and a database. A regular JDBC call returns a result set in a tabular format. This format does not directly correspond to the object-oriented data model of Java, and can complicate navigation and update operations. When a client sends a query for data through the JDBC DMS, the JDBC result set of tabular data is transformed into a DataGraph composed of related DataObjects. This enables clients to navigate through a graph to locate relevant data rather than iterating through rows of a JDBC result set.

- And more to come...

Advantages

Service Data Objects have the following advantages:

- ▶ Uniform access to data across heterogeneous sources
As already mentioned, Service Data Objects can access data from a variety of sources, including relational databases, custom data access layers, Web services, XML data stores, JMS messages, and Enterprise Information Systems.
- ▶ Becoming a standard way to implement the Business Delegate/DTO
Actually, Service Data Objects are basically Data Transfer Objects (DTOs). The Data Mediator Services are part of the specification, but they are not a standard yet. Thus, the Business Delegate with a DTO is the key pattern.
- ▶ Support for disconnected programming models
Many presentation frameworks, such as Struts or JavaServer Faces, use a disconnected usage pattern of data access. They use some kind of data transfer object, to pass application data between the layers. Service Data Objects perfectly support this model, the disconnected data objects needed are automatically generated and an optimistic concurrency model is used.
- ▶ Service Data Objects support both static and dynamic data APIs
Static data APIs are much easier to use and therefore preferred by application programmers. In some cases however, static Java interfaces for data are not sufficient, for example when it comes to dynamic queries where the shape of the resulting data is not known.
- ▶ Good tooling support available for Service Data Objects
Although Service Data Objects are very flexible, development tools can easily support them because they provide simple introspection APIs. In addition, Service Data Objects can easily be integrated into existing presentation frameworks.

Disadvantages

But there are also some disadvantages:

- ▶ Performance
Service Data Objects add another layer on top of existing persistence or data access frameworks, which on one hand increases flexibility and simplifies integration of heterogeneous data sources, but on the other hand adds some performance overhead.
- ▶ Not part of J2EE Specification

Service Data Objects are not part of any J2EE Specification. IBM and BEA Systems submitted a Java Specification Request - JSR 235 in December 2003.

► New technology

Service Data Objects were introduced in WebSphere Application Server V5.1 as WebSphere Data Objects and are now available in WebSphere Application Server V6 as Service Data Objects.

Relationship with other technologies

As mentioned earlier, SDO integrates with other data programming technologies and can use them under the covers. Table 16-1 describes how SDO relates to and integrates with some other data programming technologies:

Table 16-1 SDOs Integration capabilities

	Model	API	Data Source	Query Language
JDBC Rowset	Connected	Dynamic	Relational	SQL
JDBC Cached Rowset	Disconnected	Dynamic	Relational	SQL
Entity EJB	Connected	Static	Relational/Any (with CMP over anything ^a)	EJBQL
JDO	Connected	Static	Relational, Object	JDOQL
JCA	Disconnected	Dynamic	Undefined	Undefined
SDO	Disconnected	Both	Any	Any

a. IBM Extension - not part of the J2EE Specification

For more information about Service Data Objects:

► JSR 235: Service Data Objects

<http://www.jcp.org/en/jsr/detail?id=235>

► Introduction to Service Data Objects

<http://www.ibm.com/developerworks/java/library/j-sdo/>

16.5.2 Entity beans

Entity beans are server-side components that represent business objects stored in a persistent storage mechanism. This means, they provide an object view of transactional data in an underlying datastore that can be accessed from multiple, either local or remote, clients.

There are two different types of Entity beans available:

- ▶ Container Managed Persistence (CMP) entity beans

A Container Managed Persistence (CMP) bean is an entity bean for which the container handles the interactions between the enterprise bean and the data source. The container is responsible for synchronization of instance fields with the persistent store. When you develop a Container Managed Persistence bean, the application is insulated from the details of the persistence mechanism.

- ▶ Bean Managed Persistence (BMP) entity beans

A Bean Managed Persistence (BMP) entity bean, is simply an entity EJB where the developer manually implements the service methods to manage persistence, most notably `ejbLoad()` to load the persistent state from the backing store and `ejbStore()` to store it.

Container Managed versus Bean Managed Persistence

The decision to use Container Managed Persistence or Bean Managed Persistence entity beans is usually easy. Whenever possible, you should prefer CMP to BMP, because of the following reasons:

- ▶ Portability and flexibility

BMP entity beans contain hard-coded SQL statements that require not only a specific database layout, but may also be dependent on the database vendor. These hand-optimized SQL statements are difficult to port to another relational database or database layout. CMP entity beans on the other hand use an abstract persistence schema to specify the CMP and CMR fields in the deployment descriptor. These fields are mapped to the relational database fields during deployment. The deployment tool then generates the database vendor-specific classes. These steps ensure a high degree of flexibility and portability regardless of the used relational database schema or vendor.

- ▶ Faster development

If CMP is used, nearly all bean code can be automatically generated using actual development tools, such as IBM Rational Application Developer V6. Therefore, developers can concentrate on writing the business logic and assign the persistence and relationship management logic to the deployment tool and the EJB container. If BMP is used, the developer is responsible for loading and persisting the bean data itself.

- ▶ Performance

To write high-performance and scalable BMP entity beans, bean developers - which are usually responsible for writing the business logic - must be highly skilled in database programming as well. Unfortunately this is usually not the

case - this is the domain of database administrators, not of bean developers. Another problem is that a higher level of optimization in the bean class automatically increases the difficulty to port the bean to another database. With CMP entity beans, the deployment tool can generate highly optimized code for every specific data source. Usually the performance of CMP entity beans is by far better than corresponding BMP entity beans. For further performance increase, IBM WebSphere Application Server V6 can also generate deployed code that uses SQLJ stored procedures to access IBM DB2 UDB.

- Automatic relationship management

Bean developers can define relationships to other CMP beans in the CMP entity beans deployment descriptor. This includes cardinality, navigation, and cascade delete which are automatically generated and maintained by the EJB container. When BMP entity beans are used, relationship management, including integrity checks and navigation, has to be implemented by the bean developer.

Although CMP entity beans have overwhelming advantages over BMP entity beans, there are also some drawbacks:

- Persistence is limited to relational databases

Even though there are some approaches to extend the persistence mechanism to other than relational data sources, they are not yet available and therefore the usage of CMP entity beans is currently limited to relational databases. With BMP entity beans, access to non-relational data sources, for example using a JCA adapter, is possible.

- Reduced control over database access

With BMP, the developer has complete control over the data access logic and the persistence management of the entity bean. Therefore, the developer has the ability to call vendor-specific functions or perform complex database joins to realize huge persistence schemas. In some rare cases this high level of data access control may be required.

General coding and design guidelines for EJB entity beans

While entity beans can reduce the amount of coding work required to access persisted data, care must be taken with the use of entity beans to avoid performance problems.

- Access entity beans from session beans

Avoid accessing EJB entity beans from client or servlet code. Instead, wrap and access EJB entity beans in EJB session beans. This satisfies two performance concerns:

- When the client application accesses the entity bean directly, each getter method is a remote call. A wrapping session bean can access the entity bean locally and collect the data in a structure, which it returns by value.
- An entity bean synchronizes its state with its underlying data store at the completion of each transaction. When the client application accesses the entity bean directly, each getter method becomes a complete transaction. A store and a load follow each method. When the session bean wraps the entity bean to provide an outer transaction context, the entity bean synchronizes its state when the outer session bean reaches a transaction boundary.
- ▶ Avoid extremely fine-grained EJB models

Although local interfaces, introduced in EJB 2.0, make more fine-grained EJB models possible, take care that you do not carry the granularity to excess. Think of using dependent value classes as an alternative. They can increase performance, because no separate call and no EJB relationship is needed.
- ▶ Do not use entity EJBs to read large amounts of data

Entity EJBs are best used for manipulating small amounts of data. Returning large result sets from (default) EJB finders can be inefficient and it is often better to use JDBC directly from a session bean for this.
- ▶ Beware of significant use of EJB inheritance

In cases where entity beans significantly use ejb-inheritance, care must be taken to ensure that performance is adequate. If performance problems are encountered, they can potentially be addressed by reducing the use of inheritance in the model or by use of Bean Managed Persistence (BMP) in preference to Container Managed Persistence (CMP). Another strategy is to avoid turning each entity (table) into a single EJB - in some cases two or more related entities can be represented by a single EJB.
- ▶ Use local interfaces

If the EJB client is located in the same Java Virtual Machine as the EJB, you can take advantage of the local Interface. No network tasks are necessary and the bean parameters are passed by reference. This increases performance. So if you know that you will deploy your EJB clients in the same JVM where the EJB itself is deployed you should use the local Interface. Usually local Interfaces are used to access entity EJBs from the Facade.
- ▶ Use Caching options from EJB Container

The EJB container has three types of caching that can be performed for entity beans between transactions. Selection of the appropriate option requires an understanding of how the entity beans will be used, as there is a trade-off

between minimizing database reads and supporting Workload Management (WLM).

For more information please go to “Entity EJBs - Bean cache” on page 992.

- **Optimize EJB transaction and isolation level settings**

In EJBs we have declarative transaction management. So the developer does not need to take care of all the resources needed in the transaction, the container does. Nevertheless the definition of the transaction attributes in the deployment descriptor is an important task that can change application behavior and performance dramatically.

In WebSphere Application Server V6 a new feature called Application profiling is available that allows you to optimize transaction and isolation level settings. This feature allows you to dynamically adjust the access intent settings to the actual runtime requirements of the work currently performed. Application profiling enables you to configure multiple access intent policies on the same entity bean. It reflects the fact that different units of work have different use patterns for enlisted entities and can require different kinds of support from the server runtime environment. For more information see “Isolation levels for EJBs” on page 995 and “Application Profiling” on page 997.

Alternatives

- Java Data Objects (JDO)
- Session beans with direct access to back end

16.5.3 Java Data Objects

Java Data Objects (JDO), like Service Data Objects, have also been standardized through the Java Community Process in May 2003. With JDO, developers can easily access persistent data that can be stored in various types of back ends, such as databases, file systems or other transaction processing systems. Similar to Service Data Objects, Java Data Objects also provide a common API to simplify and unify the data access. The main difference between SDOs and JDOs however is that JDOs only solve the persistence issue, whereas SDOs use a more general approach that also includes data representation and data flow between the J2EE tiers. Compared to EJBs, most of the former advantages of JDOs more or less disappeared with the introduction of local interfaces (EJB 2.0). At the moment it is questionable if there will be further investigation into JDO 2.0 by the Java Community Process, because of the fact that it apparently overlaps with existing Java technologies and with other JSRs that are already in progress (especially EJB 3.0). Nevertheless, at the moment there are a lot of open source frameworks available that implement JDOs.

Advantages

- ▶ Universal data access for different kinds of data sources.
- ▶ Transparent persistence layer, full transaction support - like CMP EJBs.
- ▶ Good performance- even for large amounts of data.
- ▶ Lightweight technology - JDO is based on Java objects.

There is no EJB container needed for the “entity beans” itself, but you might want to implement the business logic using EJB session beans.

Disadvantages

- ▶ No built-in security (compared to EJBs).
- ▶ Not part of J2EE specification.
- ▶ Unsure future
- ▶ No built-in support for JDO in WebSphere Application Server.
- ▶ Limited tooling support available.

Alternatives

- ▶ Service Data Objects
- ▶ CMP entity beans

16.5.4 EJB session bean: direct access to back end

Reading large amounts of data with entity beans, for example just to display a large scrollable list of data, implies a big performance overhead, because a lot of transactional aware EJB objects are unnecessary instantiated from the EJB container. Therefore, in domains where a set of objects exist whose state is frequently read but very rarely updated, the usage of EJB entity beans might be overkill. In these cases, a different approach using an EJB session bean that directly accesses the back end, might be more advisable. Update operations are also possible because EJB session beans support transactions that can be automatically handled by the container.

Advantages of this approach

- ▶ High performance for large amounts of data
- ▶ Full transactional support
- ▶ Universal data access to all kinds of back ends

Most important disadvantages of this solution

- ▶ Limited portability - Implementation highly depends on data schema and kind/type of back-end system
- ▶ Not very easy to use - missing tooling support

- ▶ No caching

Alternatives

- ▶ Service Data Objects using Java DataBase Connectivity Mediator Service
- ▶ Java Data Objects

16.6 Key factors for application performance

This part discusses factors that are critical for high-performance WebSphere Application Server based applications. However, a prerequisite is to ensure that your application has a good design and architecture. The techniques and strategies outlined here will assist you in optimizing the performance of your applications. However, they will not compensate for a poorly designed or architected application. The best practices should be applied after verifying that the basic design and architecture are appropriate for the application and scalable. For application design guidelines, please start reading from the beginning of this chapter.

16.6.1 Memory

A key factor in the performance of any Java application and hence any WebSphere Application Server application is the use of memory. Unlike other programming languages, Java does not require (or even allow) programmers to explicitly allocate and reclaim memory. The Java Virtual Machine (JVM) runtime environment allocates memory when a new object is created, and reclaims the memory once there are no more references to the object. This reduces the amount of coding required, as well as minimizing the potential for memory “leaks” caused by the programmer forgetting to deallocate memory once it is no longer required. Additionally, Java does not allow pointer arithmetic. Memory deallocation is performed by a thread executing in the JVM called the garbage collector (GC). The algorithm used for garbage collection is not specified in the Java Virtual Machine specification and hence different JVM implementations may use different garbage collection algorithms.

However, all of these algorithms require the sweeping of memory to identify objects that are no longer referenced. Most also involve a compaction phase where referenced objects are copied to a particular area of memory to reduce fragmentation. More details about garbage collection can be found in “Improving Java Application Performance and Scalability by Reducing Garbage Collection Times and Sizing Memory” by Nagendra Nagarajayya and J. Steven Mayer which is available at:

<http://developers.sun.com/techttopics/mobility/midp/articles/garbage/>

An update to this article for JDK 1.4.1 is also available, it is found at:

<http://developers.sun.com/techtips/mobility/midp/articles/garbagecollection2/>

Although Java performs memory management automatically, programmers still need to be aware of the impact of memory management on performance. Creating an object consumes system resources, because the JVM must allocate memory and initialize the object. Similarly reclaiming memory using the garbage collector also uses resources, particularly CPU time. Garbage collection occurs asynchronously when free memory reaches threshold values, and it cannot be explicitly scheduled programmatically. A call to the `System.gc()` method will request that the JVM performs garbage collection. However, this is not guaranteed to happen immediately or within any specified time period.

Hence the key to minimizing the performance impact of memory management is to minimize memory usage, particularly object creation and destruction. This can be achieved by a number of means:

► Object creation

Do not create objects prematurely if there is a possibility that they will not be needed. For example, if the object is only used in one path of an if statement, then create the object inside that path rather than outside the if statement - lazy initialization. If the same object can be reused inside a loop body, then declare and instantiate it outside the loop rather than inside the loop, to avoid creating and destroying a number of objects of the same class.

► Object pools

If objects of the same class are being repeatedly created and destroyed, it can be beneficial to create an object pool that allows the objects to be reused. Classes whose objects will be used in a pool need an initializer, so that objects obtained from the pool have some known initial state. It is also important to create a well-defined interface to the pool to allow control over how it is used.

Note: IBM WebSphere Application Server V6 provides object pools for pooling application defined objects or basic JDK types. It will benefit an application which tries to squeeze every ounce of performance gain out of the system.

► Appropriate sizing for collections

Although the Java runtime environment will dynamically grow the size of collections such as `java.util.Vector` or `java.util.Hashtable`, it is more efficient if they are appropriately sized when created. Each time the collection size is increased, its size is doubled so when the collection reaches a stable

size it is likely that its actual size will be significantly greater than required. The collection only contains references to objects rather than the objects themselves, which minimizes the overallocation of memory due to this behavior.

- Temporary objects

Developers should be aware that some methods such as `toString()` methods can typically create a large number of temporary objects. Many of the objects may be created in code that you do not write yourself, such as library code that is called by the application.

- Use of static and final variables

When a value is used repeatedly and is known at compile time, it should be declared with the static and final modifiers. This will ensure that it will be substituted for the actual value by the compiler. If a value is used repeatedly but can be determined only at runtime, it can be declared as static and referenced elsewhere to ensure that only one object is created. Note that the scope of static variables is limited to the JVM. Hence if the application is cloned, care needs to be taken to ensure that static variables used in this way are initialized to the same value in each JVM. A good way of achieving this is the use of a *singleton* object. For example, an EJB initial context can be cached with a singleton using the following code fragment:

Example 16-1 Use of the singleton pattern to cache EJB initial context references

```
public class EJBHelper {
    private static javax.naming.InitialContext initialContext= null;

    public javax.naming.InitialContext getInitialContext() {
        if (initialContext == null) {
            initialContext = new javax.naming.InitialContext();
        }

        return initialContext
    }
}
```

- Object references

Although memory does not have to be explicitly deallocated, it is still possible to effectively have “memory leaks” due to references to objects being retained even though they are no longer required. These objects are commonly referred to as *loitering objects*. Object references should be cleared once they are no longer required, rather than waiting for the reference to be implicitly removed when the variable is out of scope. This allows objects to be reclaimed sooner. Care should be taken with objects in a collection, particularly if the collection is being used as a type of cache. In this case, some criteria for removing objects from the cache is required to avoid the

memory usage constantly growing. Another common source of memory leaks in Java is due to programmers not closing resources such as Java Database Connectivity (JDBC), Java Message Service (JMS) and Java Connector Architecture (JCA) resources when they are no longer required, particularly under error conditions. More information about the use of JDBC resources is provided in 16.6.4, “Database access” on page 935. It is also important that static references be explicitly cleared when no longer required, because static fields will never go out of scope. Since WebSphere Application Server applications typically run for a long time, even a small memory leak can cause the JVM to run out of free memory. An object that is referenced but no longer required may in turn refer to other objects, so that a single object reference can result in a large tree of objects which cannot be reclaimed. The profiling tool available in IBM Rational Application Developer V6, which are described in Chapter 15, “Development-side performance and analysis tools” on page 839, can help to identify memory leaks. Other tools that can be used for this purpose include Rational Purify®, Sitraka JProbe (by Quest Software), and Borland Optimizelt.

► Vertical clustering

Most current garbage collection implementations are partially single threaded (during the heap compaction phase). This causes all other program threads to stop, potentially increasing the response times experienced by users of the application. The length of each garbage collection call is dependent on numerous factors, including the heap size and number of objects in the heap. Thus as the heap grows larger, garbage collection times can increase, potentially causing erratic response times depending on whether a garbage collection occurred during a particular interaction with the server. The effect of this can be reduced by using vertical scaling and running multiple copies of the application on the same hardware. Provided that the hardware is powerful enough to support vertical scaling, this can provide two benefits: first, the JVM for each member of the cluster will only require a smaller heap, and secondly, it is likely that while one JVM is performing garbage collection, the other one will be able to service client requests as the garbage collection cycles of the JVMs are not synchronized in any way. However, any client requests that have been directed by workload management to the JVM (doing garbage collection) will be affected. Refer to 3.6, “Vertical scaling topology” on page 82 for more information about vertical clustering.

16.6.2 Synchronization

The mechanism by which access to shared resources by different threads is controlled is called *Synchronization*. While the synchronization functionality in Java is convenient and easy to use, it can introduce significant performance overhead. When a block of code is synchronized, only a single thread can

execute it at any one time. There are two performance impacts of synchronization:

- ▶ Managing the *monitors*, the objects internal to the JVM that are used to control access to synchronized resources. Although they are not explicitly accessed by programmers, there is an overhead due to the management of the monitors by the JVM.
- ▶ Reduced concurrency, since threads have to wait for other threads to exit from synchronized blocks of code.

Thus the use of synchronization should be minimized and limited to cases where it is definitely necessary. It is also good practice to clearly document all assumptions relating to synchronization, because they may not be obvious to someone reading the design or code.

When using synchronization, it is best to use specific lock objects to synchronize on. Synchronizing using the keyword can cause different methods to be unnecessarily synchronized with each other, and hence reduce concurrency. Note that synchronizing on an object has a greater overhead than calling a synchronized method. However, synchronizing the method may result in significantly greater amounts of code being synchronized, again reducing the concurrency. So the trade-off between the synchronization overhead and reduced concurrency needs to be evaluated on a case-by-case basis.

In addition to the explicit use of synchronization in application code, synchronization may be used indirectly, as some of the commonly used core Java functionality uses synchronization. Some particular examples are:

- ▶ The Java I/O libraries. It is best to minimize the use of `System.out.println()` for this reason. Use of a multithreaded logging library as discussed in 16.6.3, “Logging” on page 934 is suggested.
- ▶ Some of the Java collection classes, such as `java.util.Hashtable` and `java.util.Vector`, are synchronized. If only a single thread is accessing the data (or multiple threads are reading only), the synchronization overhead is unnecessary. Many of the newer collections introduced in Java 1.2, such as `java.util.ArrayList` are not synchronized and may provide better performance. However, care needs to be taken when accessing them from multiple threads.

16.6.3 Logging

As mentioned in 16.6.2, “Synchronization” on page 933, the Java I/O classes use synchronization. Hence `System.out.println()` should not be used for logging purposes. If a lot of output using `stdout` is generated by an application in a UNIX environment, the overhead can be avoided by redirecting `stdout` to `/dev/null` in

the production environment. However, a better approach is to use a multithreaded logging library such as the WebSphere logging facilities or Log4J (<http://jakarta.apache.org/log4j/>). In addition to providing better performance due to their multithreaded implementations, these libraries allow logging statements to be defined at a particular level, which can be dynamically changed at runtime. Thus the amount of logging in production environments can be reduced in comparison to development and test environments without requiring code changes, improving the performance of the application. It is also good practice to guard log statements so that the parameters are not evaluated if the logging level is not on. The use of guard statements is shown in Example 16-2.

Example 16-2 Use of guard statements for logging

```
if (Log.isLogging(Log.WARN) {  
    Log.log(LOG.WARN, "This is a warning");  
}
```

16.6.4 Database access

The Java Database Connectivity (JDBC) API provides a vendor-independent mechanism to access relational databases from Java. However, obtaining and closing a connection to a database can be a relatively expensive exercise, so the concept of *connection pools* has been introduced. When a database operation is to be performed, a connection can be obtained from the pool, which contains a defined number of connections to the database that have already been established. When the connection is closed, it is returned to the pool and made available for reuse. Using connection pooling can significantly reduce the overhead of obtaining a database connection. However, the connection pool is accessed via a *datasource*. References to the *datasource* are obtained by performing a lookup via the Java Naming and Directory Interface (JNDI). This lookup is an expensive operation, so it is good practice to perform the lookup once and cache the result for reuse.

JDBC resources should always be released once they are no longer required. This includes `java.sql.ResultSet`, `java.sql.Statement` and `java.sql.Connection` objects, which should be closed in that order. The code to close the resources should be placed in a finally block to ensure that it is executed even when an exception condition occurs. Failure to properly close resources can cause memory leaks, and can cause slow response due to threads having to wait for a connection to become available from the pool. Since database connections in the pool are a limited resource, they should be returned to the pool once they are no longer required.

If an application repeatedly executes the same query, but with different input parameters, then performance can be improved by using a `java.sql.PreparedStatement` instead of `java.sql.Statement`.

Turning off auto commit for read only operations may also increase performance.

To avoid having to retrieve and process large amounts of data, sometimes it is beneficial to use database stored procedures for implementing some of the application logic. Alternatively, in some cases calls to the database can be minimized by using a single statement that returns multiple result sets.

There are different types of JDBC drivers available, some written in pure Java and others that are native. Although use of a native driver can reduce portability, performance may be better with a native driver.

16.7 General coding issues

This section describes a variety of techniques to improve performance of WebSphere applications, particularly through the efficient use of the core Java functionality.

- ▶ Although strings are a simple and efficient data structure in many languages such as C/C++, there is overhead associated with the use of strings (`java.lang.String`) in Java. Java strings are immutable; once created their value cannot be changed. Hence operations such as string concatenation (+) involve the creation of new strings with the data copied from the original strings, creating more work for the garbage collector as well. When performing string manipulation operations, use of `java.lang.StringBuffer` as an alternative to `java.lang.String` can improve performance.
- ▶ Although the reflection facilities in Java can be extremely useful and allow for elegant implementations, reflection is an expensive operation that should not be used indiscriminately. This is another case of where a trade-off between performance and elegance of the solution needs to be made.
- ▶ Avoid creating excessively complicated class structures. There is a performance overhead in loading and instantiating these classes.
- ▶ Avoid excessive and repeated casting. Once an object has been cast, assign a variable of the correct type and reuse this reference.
- ▶ Use of “?:”, where the equivalent if blocks simply assign one value or another, provides better performance for most JVMs.
- ▶ When iterating n items, iterating from n-1 to 0 instead of 1 to n is quicker for most JVMs. See Example 16-3 on page 937.

Example 16-3 Iterating through a loop *n* times

```
for (int=n-1;i>0;i--) {  
    // Do something in a loop.....  
}
```

- ▶ Avoid repeatedly calling the same method within a loop if the result is the same every time. Instead store the value in a variable prior to entering the loop and use this stored value for each loop iteration.
- ▶ Use `System.arraycopy()` to copy the contents of one array to another, rather than iterating across each array element and copying it individually.
- ▶ Where possible, declare methods with the `final` modifier if they will not be overridden. Final methods can be optimized by the compiler by method in-lining. The byte code of the method is included directly in the calling method. This avoids the overhead of performing a method call.
- ▶ When reading and writing small amounts of data, use of the Java Buffered I/O classes can significantly improve performance, by minimizing the number of actual I/O calls that need to be made.
- ▶ Avoid the overuse of exceptions. Throwing and catching exceptions can be expensive. Exceptions should be mainly used for (infrequent) error conditions, not as a normal mechanism for control flow by the application. Although throwing and catching exceptions should be minimized, checking for them with the use of `try {} catch {}` blocks is not particularly expensive so this can be used extensively if required. However, avoid catching an exception so that it can be simply rethrown and caught again later. Catch the exception at the level where it will be handled. Also beware of printing stack traces for exceptions, because they can be quite large.
- ▶ It is best to avoid spawning of new threads. Spawned threads do not have access to J2EE resources such as JNDI, security or transaction contexts. Rather than spawning a thread to act as a server to receive incoming messages, consider using message-driven beans (MDBs).
- ▶ In many applications, performance can be improved by performing some caching of data by the application. Note that if this is done, consideration must be given to periodically flushing the cache to avoid it growing continuously. Also be careful with making assumptions about requests for a client always being served by a particular application server instance. Even if session affinity is used, in a failover situation, HTTP requests can be serviced by a different application server instance, which may not have the cached data. We recommend that the cache be implemented using a well-defined interface, and that data that is not in the cache be retrieved again, transparent to the rest of the application.

Note: You can use the WebSphere Dynamic Cache service to intercept calls to cacheable objects and store their output in a Dynamic Cache. Refer to Chapter 10, “Dynamic caching” on page 501 for further information about this topic or to the content related to dynamic caching in the WebSphere InfoCenter at

http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp?topic=/com.ibm.websphere.base.doc/info/aes/ae/tdyn_dynamiccache.html

16.8 Reference

For more information, see the following:

- ▶ IBM developerWorks at:

<http://www.ibm.com/developerworks/>

<http://www.software.ibm.com/wsdd/>

- ▶ White paper “WebSphere Application Server Development Best Practices for Performance and Scalability”, by Harvey W Gunther, 7 September 2000 found at:

http://www.ibm.com/software/webservers/appserv/ws_bestpractices.pdf

- ▶ Java Performance Tuning Web site at:

<http://www.javaperformancetuning.com/>

- ▶ Stacy Joines, et.al., *Performance Analysis for Java Web Sites*, Addison-Wesley, September 2002, ISBN 0201844540

Performance tuning

This chapter discusses tuning an existing environment for performance. While it does not provide specific tuning values, which would be unique to each environment and application, this chapter provides a guide for testing application servers and components of the total serving environment that should be examined and potentially tuned to increase the performance of the application.

Before reading this chapter, please take the time to review Chapter 2, “Infrastructure planning and design” on page 39.

This chapter consists of four main parts:

- ▶ “Testing the performance of an application” on page 940
- ▶ “Tools of the trade” on page 945
- ▶ “Performance monitoring guidelines” on page 965
- ▶ “Performance tuning guidelines” on page 975

17.1 Testing the performance of an application

Application performance testing is an important component of the software deployment cycle. It becomes even more important with respect to Web applications, since an end user's tolerance for slow applications is generally much lower than that of a captive internal audience. Performance testing has been a subject of many products and white papers recently, and has spawned a new IT industry dedicated to Web application testing.

17.1.1 Introduction to application performance testing

When performance testing a Web application, several requirements must be determined either through interpretation of data from an existing application that performs similar work, or from best-guess estimates. Those requirements are:

- ▶ Average request rate
What is the expected number of users who will access this application? This is generally expressed in hits per month, day, hour, or minute, depending on volumes. This should be re-evaluated regularly.
- ▶ Peak request rate
How many pages will need to be served per second? This also should be re-evaluated regularly.
- ▶ Average concurrent users
What is the average number of users accessing the application at the same time during regular usage hours? This should be planned for, expected, and re-evaluated on a regular basis.
- ▶ Peak concurrent users
This is the maximum number of concurrent users that will visit your site during peak time.
- ▶ Regular usage hours
This value defines your off-peak hours. This is required to simulate a realistic workload.
- ▶ Peak usage hours
During this time, most of your traffic will happen and a performance degradation would impact most of your users.
- ▶ Site abandonment rate
How long will a user stay on your page before he leaves the site or closes the browser?

The user base, especially for Web applications, is a difficult number to determine, especially if this is an application that is performing a new function on a Web site. Use existing Web server measurements to provide a “best-guess” number based on current traffic patterns for the site. If capturing these numbers is not possible, then the best option is to determine the breaking point for the application within the intended deployment infrastructure. This provides the ability to monitor once the application is live and to provide increased capacity prior to a negative user response due to load.

When dealing specifically with applications running in WebSphere Application Server, there are performance testing protocols that can be followed. A good example of this can be found in the article “Performance Testing Protocol for WebSphere Application Server-based Applications” by Alexandre Polozoff, found at:

http://www.software.ibm.com/wsdd/techjournal/0211_polozoff/polozoff.html

One important point to remember is that performance tuning is more of an “art” than a science. Don’t worry if you get the impression that you will not be able to achieve your tuning goals if you believe you lack the “talent” for that art, because, on the other hand, performance tuning should also be seen as an art that strictly follows the recurring, monotonous trifold process of Testing - Evaluating - Tuning. This process requires that you know your system, your environment and application very well, that you know what you want to test, what goal(s) you want to achieve, and also that you are familiar with the tools you are going to use for load testing, all of which implies more of a solid handicraft than an artist’s creative work. But having a bit of intuition and developing a feeling for your work will most certainly help you in finding the best configuration. Finally, experience will be your most valuable friend, so get to know different load testing tools on various environments to gain comprehensive, in-depth knowledge.

It is important to keep in mind that it is impossible to make up for poor application design or application code by tuning WebSphere. It is also important to remember that performance tuning is an ongoing process. Continual reviews of performance should be done to ensure that changes in load, application behavior, and site behavior have not adversely impacted application performance. Also, there are no hard rules for performance tuning. What may be appropriate tuning for one application may not be appropriate for another. It is more important to understand the concepts associated with tuning, and make adjustments based on the understanding gained from those concepts.

Important: Integrate your developers into the performance tuning life cycle. Doing this ensures that your application code is tuned before tuning WebSphere, since the biggest impact on performance can be made by tuning your application code.

17.2 Selecting testing tools

Deploying applications that perform and scale in an acceptable manner is not an accidental occurrence. Producing high performance software requires that you include several rounds of stress testing during the development cycle. You can use one or more of the many open source or commercial stress testing tools to automate the execution of your stress tests.

The primary purpose of stress testing tools is to discover under what conditions your application's performance becomes unacceptable. You do this by changing the application inputs to place a heavier and heavier load on the application and measuring how performance changes with the variation in those inputs. This activity is also called load testing. However, load testing usually describes a very specific type of stress testing: increasing the number of users to stress test your application.

The simplest way to stress test an application is to manually vary the inputs (for example the number of clients, size of requests, frequency of requests, mix of requests) and then chart how the performance varies. If you have many inputs, or a large range of values over which to vary those inputs, you probably need an automated stress testing tool. Moreover, you will want test automation to repeat test runs following environmental or application-specific changes once you uncover an issue.

Important: We recommend making only one change at a time in between test runs so the results are measurable.

If you are testing manually, it can be difficult to accurately reproduce an identical set of tests across multiple test executions. When it comes to having multiple users testing your application, it is almost impossible to run manual tests consistently and it can be very difficult to scale up the number of users testing the application.

Today, there is no generic, one-size-fits-all stress testing tool. Every application differs in what inputs it takes and how it executes them. Java and WebSphere-based Web applications generally receive requests from clients via the HTTP protocol. There are many stress testing tools that can simulate user activity over HTTP in a controlled and reproducible manner.

With so many stress testing tools available today, how can you choose the one that is most appropriate for your application? Some of the points to consider when evaluating stress testing tools include:

- ▶ Client interaction

The stress testing tool must be able to handle the features and protocols that your application uses.

- ▶ Simulation of multiple clients

This is the most basic functionality of a stress testing tool.

- ▶ Scripted execution with the ability to edit scripts

If you cannot script the interaction between the client and the server, then you cannot handle anything except the most simple client requests. The ability to edit the scripts is essential; minor changes should not require you to go through the process of re-generating a script.

- ▶ Session support

If a stress testing tool does not support sessions or cookies, it is not very useful and may not be able to stress test Java and WebSphere applications.

- ▶ Configurable numbers of users

The stress testing tool should let you specify how many simulated users are running each script or set of tasks, including allowing you to vary the number of simulated users over time. Many stress testing tools enable you to start with a small number of users and ramp up slowly to a higher numbers of users.

- ▶ Reporting: success, errors, and failures

The tool you choose must have a defined way to identify a successful interaction, as well as failure and error conditions. An error might be getting no Web page back at all, whereas a failure might be getting the wrong data back on the page.

- ▶ Page display and playback

A useful feature in many stress testing tools is the capability to inspect some of the pages that are being sent to the simulated users or to replay entire test scripts. You can then be confident that the stress test is functioning as you expect.

- ▶ Exporting test results

After running a stress test, you may want to be able to analyze the test results using various tools that are external to the stress testing tool, including spreadsheets and custom analysis scripts. Most stress testing tools include extensive built-in analysis functions, but being able to export the data gives you more flexibility to analyze and catalog the data in arbitrary ways.

- ▶ Think time

Real-world users do not request one Web page immediately after another. There are generally delays between viewing one Web page and the next. The

term *think time* is the standard way of expressing the addition of a delay into a test script to more realistically simulate user behavior. Many stress testing tools support randomly generated think times based on a statistical distribution.

- ▶ Variable data

Live users do not work with the same set of data on each interaction with your application. During a stress test, this should also be true of your simulated users. It is easier to make your simulated users appear to be working with varied data if the stress testing tool supports data input from lists, files, or a database.

- ▶ Script recording

Rather than writing scripts, it is much easier to manually run through a session with your browser and have that session recorded for later editing. Most stress testing tools include provisions for capturing manual interaction with your application.

- ▶ Analysis tools

Measuring performance is only half the story. The other, and perhaps more important, half of stress testing is analyzing the performance data. The type of analysis tools and degree of detailed analysis you can perform depends directly on what analysis tools are supported by the tool you select. Therefore, evaluate this support in the tools you are considering carefully.

- ▶ Load distribution

Your deployed application may well need to support hundreds of concurrent users once in production. How can you simulate this level of traffic in a stress testing environment? A typical workstation running a stress testing tool will likely begin bottlenecking once approximately 200 virtual users are running. To simulate a greater number of users, you can distribute the stress testing load across multiple workstations. Many of the available stress testing tools support distribution of load and you will certainly want this feature for large scale stress testing.

- ▶ Measuring server-side statistics

The basic stress testing tool measurement is client-based response times from client/server interactions. However, you may also want to gather other statistics, such as the CPU utilization or page faulting rates. With this server-side data, you can then do useful things like view client response times in the context of server load and throughput statistics.

17.3 Tools of the trade

The following is an overview of a small array of load testing tools. It is by no means complete, nor does it imply a recommendation or endorsement of these tools. The most comprehensive performance testing tools are (in no special order) Rational Performance Tester 6.1, Segue SilkPerformer, and Mercury LoadRunner. These tools have a very broad and deep functionality but they are also quite expensive, while for your special case some other (possibly open-source licensable) tool might do the job as well. Sections 17.3.1, “ApacheBench” on page 945 and 17.3.2, “OpenSTA” on page 948 give a brief introduction to these two open-source load testing packages. 17.3.3, “Rational Performance Tester” on page 956 gives an overview of this product and in 17.3.4, “Other testing tools” on page 964, you will find a few links to additional performance testing products and projects.

17.3.1 ApacheBench

ApacheBench (AB) is a very simple tool for quickly generating HTTP GET and POST requests. It is part of the Apache HTTP Server and the IBM HTTP Server powered by Apache. It is automatically installed with IBM HTTP Server and the executable can be found in <IHS_HOME>\bin\ab. For a manual and list of options, refer to:

<http://httpd.apache.org/docs/programs/ab.html>

Note: If you are not using a Web server that includes ApacheBench, you can always download the Apache HTTP Server source-code from the following Web site and compile the AB utility yourself:

<http://httpd.apache.org/download.cgi>

ApacheBench was mainly designed as a benchmarking tool, and it is only usable for very basic load testing because of several limitations:

- ▶ It has no functionality to record a browser click stream and requests have to be specified on the command line.
- ▶ Although the HTTP POST method is supported, POST request data has to be put into a file before the request is sent.
- ▶ There is no support for distributed load generation.
- ▶ It cannot be used for scripting of scenarios or testcases, and can test only one URL at a time.
- ▶ It cannot retrieve cookie data from the response, although cookie information can be specified on the command line, making session-aware performance testing possible in a very limited way.

- There is no ramp-up or ramp-down support; ApacheBench always starts at the specified load.

The ramp-up period is the time before all virtual testers are active and the ramp-down period is the time where all virtual testers that started during ramp-up are stopped and are no longer generating load on the system simultaneously. This is demonstrated in the graph in Figure 17-1.

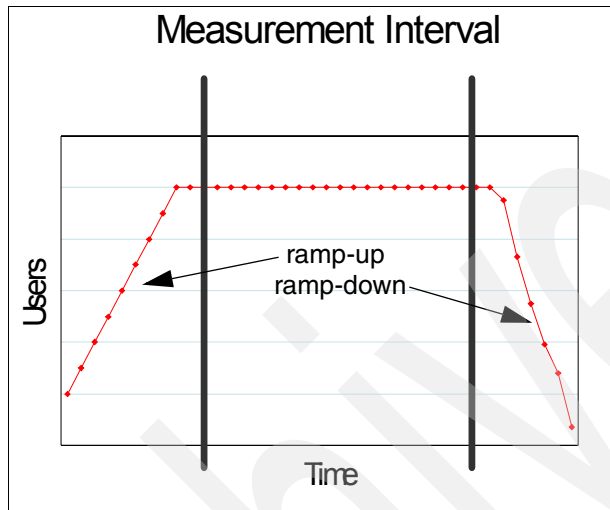


Figure 17-1 Ramp-up and ramp-down period demonstration

- ApacheBench does not simulate users, but repeated concurrent requests. This means that you have to use caution when specifying the number of clients you want to test with.

Nevertheless, AB has several advantages. It is free, very easy and quick to use, and there is no additional setup required since it is part of IBM HTTP Server. It can be useful for very simple and ad hoc performance tests, where the goal is, for example, to stress a Web server or application server to get a rough idea of how many requests (of one kind) can be processed per second. Another use can be to stress the application server, creating a multitude of HTTP sessions at a time, while using Tivoli Performance Viewer to monitor the servers' performance. The example on page 948 shows a sample ApacheBench command to stress test the Trade 6 Web primitive PingJSP.

Important: Be careful when selecting the URL you wish to test with AB. Since you can only specify one URL at a time as the input parameter, make sure to select the URL you really mean to test!

Sometimes, a Web page (in our example, the Trade 6 main URL `http://<your_host>/trade/`) will provide a detailed display view which is generated in two steps:

1. First, the browser retrieves the response from the server which only contains a frameset or a redirect.
2. Then the browser creates additional requests to receive the actual content, such as images and so forth.

ApacheBench is by no means a browser and does not know about framesets, links, images, or dynamic content like Javascript; it will not follow any redirects or even resolve simple HTML `` tags to retrieve images. This behavior will certainly falsify your performance results if you are not careful. Following our Trade 6 example, be sure to use a request URL that returns meaningful data (not just redirects or framesets) from the application server.

For example, use:

- ▶ `http://<your_host>/trade/PingJsp.jsp`
- ▶ `http://<your_host>/trade/scenario`

Command line options

The most common and useful options for AB are the following:

- | | |
|-------------------------|--|
| -h | Displays help and usage. |
| -n requests | The (absolute) number of requests to perform for the benchmarking session. The default is 1. Keep in mind that AB does not implement <i>think time</i> ! If you set the number of requests too high then you will effectively perform a denial-of-service attack on your server. |
| -c concurrency | The number of simultaneous requests to perform (virtual users). The default is 1, which means no concurrency. |
| -v level | Used to set the verbosity level. Level 2 and higher will print HTTP headers and the HTTP response body. |
| -A username:pass | Used to supply basic authentication credentials to the server. |
| -C name=value | Add a cookie-header with name and value to the requested object. |

Note: The number of requests is not set per concurrent user. If you specify `-n 20` and `-c 10`, each of the 10 virtual users will only perform two requests.

As an example, the command for stress testing the Trade 6 primitive PingJSP using ApacheBench is:

```
C:\IHS\bin> ab -g test -c 5 -n 50  
"http://http1.itso.ibm.com/trade/PingJsp.jsp"
```

In this example, we perform 50 requests using five virtual users, that is, each of the virtual users performs 10 requests.

The results of an ApacheBench stress test returns values such as:

- ▶ Time taken for tests
- ▶ Requests per second
- ▶ Time per request (mean)
- ▶ Time per request (mean, across all concurrent requests)
- ▶ Transfer rate (Kbytes/sec)
- ▶ Percentage of the requests served within a certain time

17.3.2 OpenSTA

Open System Testing Architecture (OpenSTA) is open source software licensed under the GNU General Public License. This CORBA-based distributed software testing architecture allows you to simulate the activity of hundreds to thousands of virtual users and thus generate heavy loads on your (test) systems.

Using OpenSTA, you can gather information on response times and resource utilization from all server systems included in the load test (such as Web servers, application servers, database servers, etc.) and use it for analysis.

Feature overview

The name OpenSTA stands for a collection of tools that build on the distributed architecture. These tools allow you to perform scripted HTTP load or stress tests including performance measurements run from a single machine or distributed and coordinated across many machines.

The software package can be obtained from <http://opensta.org/>. There is also an excellent community site for additional information and documentation to be found on <http://portal.opensta.org/>.

The source code can be obtained from:

<http://opensta.sourceforge.net/>

The current version is OpenSTA 1.4.2 and contains the following features:

- ▶ Intelligent script recording: automatic cookie-based session support in recorded scripts

- ▶ Script modeling in a BASIC like Script Control Language (SCL)
- ▶ Script debugging using single-step mode and breakpoints
- ▶ Modular test-suite creation
- ▶ Supported protocols: HTTP 1.0, HTTP 1.1, HTTPS
- ▶ Single machine based load generation
- ▶ Distributed load generating across multiple machines with one controller
- ▶ Support for controlling load generation over the Internet
- ▶ Extensive data collection and simple statistics graphs provided
- ▶ Collected data can be exported into comma separated text files
- ▶ Additional performance measurement data collected through Simple Network Management Protocol (SNMP) and Windows NT performance monitor
- ▶ Tutorial, user's guide and detailed online help available
- ▶ Online community and Web-based discussion forum; for details, see:
<http://portal.opensta.org/>
- ▶ Commercial support and services available
- ▶ Supported load generating platforms: Windows NT and Windows 2000
- ▶ Easy to use, but more powerful features require manual script modelling (for example, form-based login)

Prerequisites and installation

We were using the current OpenSTA version 1.4.2 to test our sample topology described in Chapter 8, "Implementing the sample topology" on page 387. The listed requirements are for that version only. See the latest release notes and the download package for the most up-to-date system requirements, found at:

<http://www.opensta.org/download.html>

Software requirements

- ▶ Windows 2000
- ▶ Windows NT with service pack 5
- ▶ Microsoft Data Access Components (MDAC) version 2.5 (minimum; this can be obtained from <http://microsoft.com/data/download.htm>)

Supported Web browsers for script recording

- ▶ Microsoft Internet Explorer 4, 5, and 6
- ▶ Netscape Navigator 4.7

Note: Other browsers (for example, Opera, etc.) can be used for script recording, but their proxy settings have to be configured manually instead of being configured automatically by the OpenSTA Script Modeler.

Installation

Install OpenSTA by running the setup wizard and following on-screen directions.

The OpenSTA architecture

OpenSTA supplies a distributed software testing architecture based on CORBA, which enables you to create and run tests across a network. For a detailed overview of the architecture and its components, refer to the *Getting Started Guide* and the *User's Guide* in the documentation section of <http://opensta.org>.

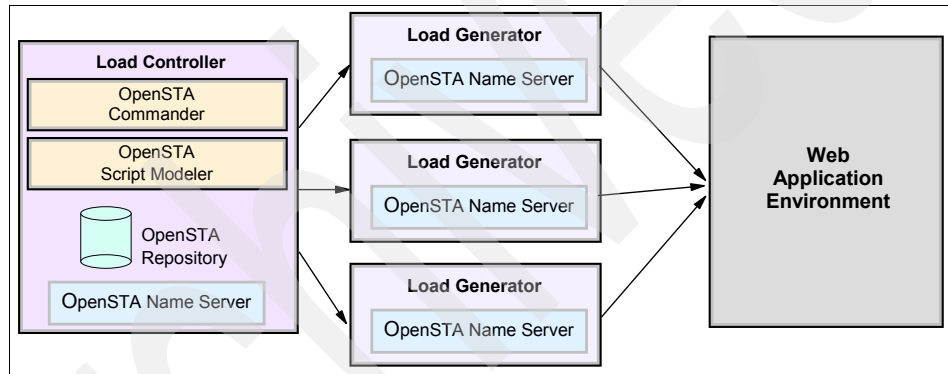


Figure 17-2 Overview of the OpenSTA distributed testing architecture

OpenSTA Name Server

The *OpenSTA Name Server* configuration utility is the component that allows you to control your test environment. After installation, you will notice the OpenSTA Name Server running, which is indicated by an icon in the Windows task bar. For a distributed test, every load generating node has to have the OpenSTA Name Server installed and configured to point to the controlling node, also called the *repository host*.

OpenSTA Commander

The *Commander* is the graphical user interface that functions as the front end for all test development and test coordination activity. This is the main component of OpenSTA you need to run in order to perform load tests. In the left pane, it shows a view of the OpenSTA *repository*, containing all defined tests, scripts and collectors.

OpenSTA Script Modeler

The *Script Modeler* is the recording and script developing environment of OpenSTA. It is launched through the Commander by double-clicking a script name in the repository tree window.

Recording the Trade 6 script

With the Script Modeler, you can record and/or develop and refine your scripts. They form the content of your tests and let you generate the desired Web activity during a load test. They are stored in the repository from where they can be included into multiple tests. Perform these steps to record a new script:

1. Create a new, empty script by selecting **File -> New Script -> HTTP**, or by right-clicking the **scripts** node in the repository tree window and selecting **New Script -> HTTP** from the pop-up menu. Enter the name TRADE6SCENARIO.
2. Launch the Script Modeler by double-clicking the previously created script.
3. Select **Options -> Browsers** and select the browser you want to capture with.
4. Select **Capture -> Record** or alternatively click the record icon (the red circle) in the toolbar to start recording. The settings that allow automatic cookie handling are enabled by default. Refer to the online help for details about recording options.

Note: OpenSTA provides a proxy gateway that intercepts all HTTP traffic during recording. Script Modeler will automatically configure your browser's proxy setting to point to that gateway, and reset it to the previous values after recording. By default, the proxy gateway uses port 81 on the local host.

5. After the browser window has launched, open the URL:

`http://<host_name>/trade/scenario`

In our case:

`http://http1.itso.ibm.com/trade/scenario`

Do not reload the page, because we only want to capture exactly one request. When the page has loaded completely, either close the browser window, or switch back to the Script Modeler and select **Capture -> Stop** from the menu; both methods will terminate your recording session.

Tip: If you want to start your script with a certain page within your Web application, you should first navigate to that page. The Script Modeler window offers a Pause button on the toolbar which allows you to temporarily stop recording. Follow these steps:

1. Open the browser window and launch your home page.
2. Click the **Pause** button.
3. Navigate to the page with which you want to start your script.
4. Resume recording by clicking the **Pause** button again.

5. Remove all references to cascading style sheets or image files from your script, because we only want to simulate load using the servlet provided by Trade 6 for this purpose.

Note: We are only removing the cascading style sheets and image files because these are not relevant for our test scenario. This is because the servlet that we are calling will generate load and simulate a test scenario itself. This is a special case. If you want to simulate user behavior, you usually do not want to perform this step!

6. Save your script and exit Script Modeler.

You have now created a simple user session in the Trade 6 application. An excerpt from that script is shown in Example 17-1. You can now view, replay and modify the recorded script inside Script Modeler. If you change the script manually, use the compile icon on the toolbar to find syntax errors.

Example 17-1 An excerpt of the recorded OpenSTA script "TRADE6SCENARIO"

```
Start Timer T_TRADE6SCENARIO
PRIMARY GET URI "http://http1.itso.ibm.com/trade/scenario HTTP/1.0" ON 1&
    HEADER DEFAULT_HEADERS&
    ,WITH {"Accept: image/gif, image/x-bitmap, image/jpeg, image/pjpeg,
application/vnd.ms" &
        "-excel, application/vnd.ms-powerpoint, application/msword, */*",&
        "Accept-Language: en-us",&
        "Connection: Keep-Alive"}

Load Response_Info Header on 1&
    Into cookie_1_0&
    ,WITH "Set-Cookie,JSESSIONID"

WAIT 2494

DISCONNECT FROM 2
```

DISCONNECT FROM 1

SYNCHRONIZE REQUESTS

End Timer T_TRADE6SCENARIO

Randomizing request using variables

A randomization of the requests is quite easily achieved using variables. All you need is the script editor and a little knowledge of the OpenSTA scripting language. You could, for example, insert two variables (user and password information) that are filled with different account data on each request, and build a new HTTP GET string using those variables. The variable information can be read by the scripting engine from a flat file, effectively simulating different, concurrently active user sessions.

Additional Script Modeler features

Following is a selection of additional, very useful script modelling features:

- ▶ **Custom timers:** To evaluate the response time for business transactions, manual timers can be inserted into the scripting code.
- ▶ **Output stream parsing:** Verification of the response header and body can be done by manually parsing the output stream using the OpenSTA scripting language.

Performance testing using the OpenSTA Commander

Using the Commander, you can define tests; each test can consist of one or more *task groups*, which are sequences of test scripts. Each task group has to contain at least one script. Once a script has been associated with a task group, it is called a *task*.

Test execution settings like the number of iterations or the number of virtual users (VU) can be specified per task group, while the number of iterations per task is also configurable. This modular structure makes the scenario easy to implement, where a user logs in first, then uses the Trade 6 application a configurable number of times, and finally logs out (see Example 17-2 on page 954).

Test Trade6 will be repeated 10 times, consisting of:

```
|  
|--Task group Trade6_1: 50 VU  
|   |--Task-1: T6_Login, repeat count: 1  
|   |--Task-2: T6_Scenario-1, repeat count: 500  
|   |--Task-3: T6_Scenario-2, repeat count: 250  
|   |--Task-4: T6_Logout, repeat count: 1
```

To run a performance test using OpenSTA, perform these steps:

1. Create the test
2. Specify runtime parameters
3. Execute the test
4. View the results

Create a test

The first step is to create a new test and add the previously recorded script:

1. In the Commander window, select **File -> New Test -> Tests** and enter TRADE6 as the name of the new test.
2. Drag the script TRADE6SCENARIO from the repository view on the left side onto the Task1 column on the right side of the Commander window. This creates a new task group called TRADE6_1.

Specify runtime parameters

The next step is to configure the number of iterations for the task and the number of virtual users that will concurrently perform the scripted task. In this example, we will use 40 virtual users performing 10 iterations of the TRADE6SCENARIO task.

1. In Column Task1, select **TRADE6SCENARIO**. Change the iterations field to 10.
2. Click the field containing 1 in column VUs. Enter 40 for the Total number of virtual users for this task group.

Additional OpenSTA Commander features

Collectors, which gather operating system performance data like CPU utilization, network and disk I/O, etc., can be defined in the OpenSTA Commander. These collectors will gather data during the test execution and can then be correlated with the actual test results. The two supported collector schemes to gather data from are:

- Performance data available from the Windows System Monitor (Windows Management Instrumentation WMI)

- ▶ Simple Network Management Protocol (SNMP) data from SNMP-enabled hosts or devices

Execute a test

The final step is to start the test using the OpenSTA Commander; while it is executing, the test's progress and statistics can be viewed. First, you have to change the monitoring interval to five seconds. That way, the statistics will be updated every five seconds (in a real world scenario, the monitoring interval should be set even higher so as to not influence the performance test by wasting CPU and network resources used for collecting performance data).

1. Set the monitoring interval to five seconds:
 - a. Select the **Monitoring** tab.
 - b. Click the **Tools** symbol.
 - c. Set both Task Monitoring Interval and Sampling Frequency to 5.
2. To start the test, select **Test -> Execute Test** or click the green **Play** icon on the toolbar. During the test, you can watch the progress and test statistics:
 - a. Switch to the monitoring view by selecting the **Monitoring** register.
 - b. Enable the **Summary** checkbox on the right side of the Commander window to see the following current statistics:
 - Active virtual users
 - Test duration
 - HTTP requests/second
 - Number of successful/failed HTTP requests

Right-click inside the **Summary** window to select and deselect additional statistics data.

The test will stop automatically after all 40 VUs have cycled through their 10 iterations of the TRADE6SCENARIO task.

View results

After the test has stopped and results data has been collected, switch to the Results view tab to examine the test data and statistics. There you can view different reports and even some graphs.

The most useful reports and graphs are:

- ▶ Test Summary, Audit Log, Report Log, and Test Error Log
- ▶ HTTP Data List (for debugging HTTP responses)
- ▶ HTTP Response Time (Average per Second) versus Number of Responses Graph

This graph displays the average response time for requests grouped by the number of requests per second during a test run.

- ▶ HTTP Errors versus Active Users Graph

This graph displays the effect on performance measured by the number of HTTP server errors returned as the number of active Virtual Users varies during a test run.

- ▶ HTTP Responses versus Elapsed Time Graph

This graph displays the total number of HTTP responses per second during the test execution.

- ▶ HTTP Response Time versus Elapsed Time Graph

This graph displays the average response time per second of all the requests issued during the test run.

- ▶ Timer Values versus Active Users Graph

- ▶ Timer Values versus Elapsed Time Graph

OpenSTA Commander also provides two additional features to help you with further data analysis:

- ▶ Exporting results data

This feature allows you to export every text-based report into a comma-separated-values (CSV) file. Every graphics report can be exported directly into Microsoft Excel, where you can perform additional statistical calculations. Export the data by right-clicking inside the opened report window; select **Export** and save the CSV-file or open it inside Excel.

- ▶ URL filtering the report data

This feature is available when right-clicking inside a graphics report and selecting **Filter URLs**. For example, you can filter out any URLs so that only the servlet request to /trade/scenario is left. Then the graph displays only the response times for the actual dynamic content of your load test.

The most interesting reports are:

- ▶ HTTP Data List
- ▶ HTTP Response Time versus Elapsed Time
- ▶ HTTP Responses versus Elapsed Time

17.3.3 Rational Performance Tester

IBM Rational Performance Tester 6.1 (RPT 6.1) is a multi-user system performance test product hosted in the Eclipse shell with a Java-based execution

engine. The focus of IBM Rational Performance Tester 6.1 is multi-user testing of Web applications.

IBM Rational Performance Tester 6.1 is being built from an entirely new code base running in the Eclipse shell; its predecessor, IBM Rational Performance Tester 6.0, is a Win32-based product whose primary components are Rational Robot and Rational TestManager.

IBM Rational Performance Tester 6.1 provides major improvements in the areas of ease-of-use and scalability.

Important: Content in this section is based on a Beta version of RPT 6.1.

This section gives you a short introduction to the functions of this product, in particular how to record and run a performance test. We do not compare it (feature-wise) to other products in this chapter, since the suitability of any of these products depends highly on your requirements. For more information about this product, please contact your IBM representative or go to the following Web site:

<http://www.ibm.com/software/awdtools/tester/performance/>

This Eclipse-based version of Rational Performance Tester is available for Windows and Linux (RedHat Enterprise Linux WS and SUSE Linux Enterprise Server) platforms.

Good performance of multi-user Web applications is a necessity, not a luxury. Even small performance deficiencies and scalability failures can slow business to a halt or cause customers to take their business elsewhere. In order to capture performance problems before deployment, software development and test teams must proactively measure the ability of their applications to rapidly and accurately support multiple, concurrent users. IBM Rational Performance Tester was built to address this need.

IBM Rational Performance Tester is a load and performance testing solution for teams concerned about the scalability of their Web-based applications. Combining ease-of-use features with flexibility, Rational Performance Tester simplifies the test creation, execution and data analysis to help teams ensure the ability of their applications to accommodate required user loads before the applications are deployed.

The steps to perform a simple load test with IBM Rational Performance Tester are:

1. Record a Performance Test
2. Creating a Performance Schedule

3. Run a Performance Schedule

Record a Performance Test

Before we can run any tests, we first have to create our test project in the workspace and record a Performance Test case. To do this:

1. Open your IBM Rational Software Development Platform. After it has started, it should look like the one in Figure 17-3.

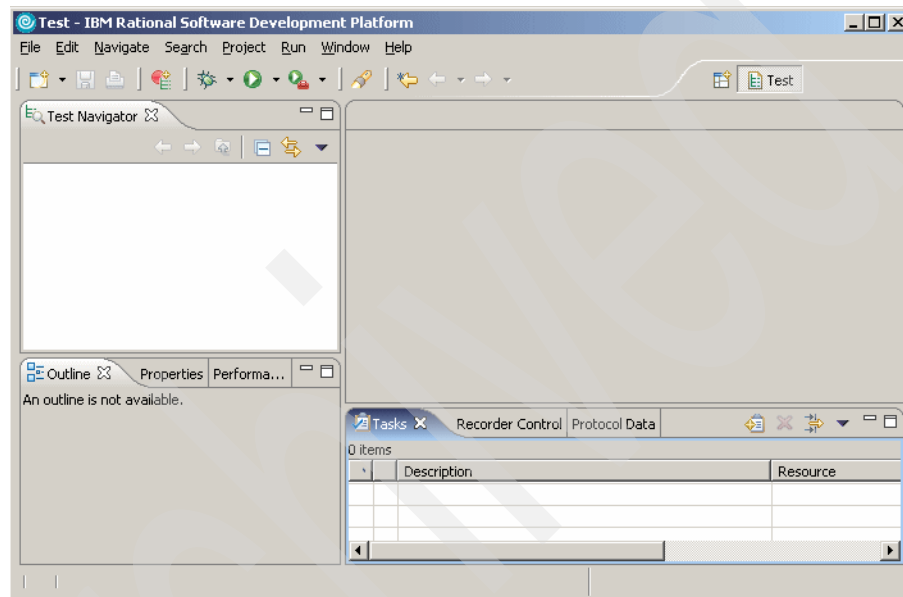


Figure 17-3 Rational Software Development Platform workspace

2. Click **File** -> **New** -> **Performance Test Project** to create a new test project called SAW404-Trade6 in your workspace and click **Finish**. See Figure 17-4 on page 959.

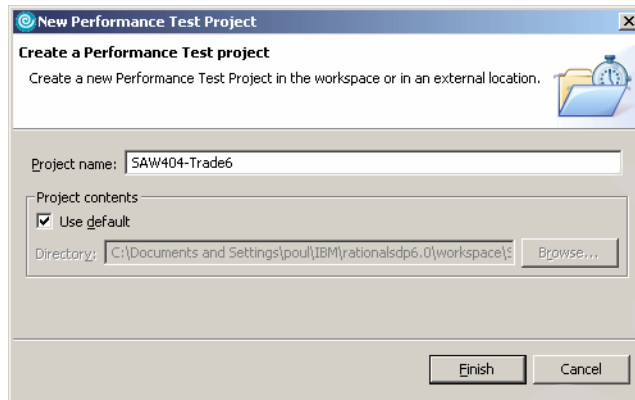


Figure 17-4 Create new performance test project wizard

3. Click **Yes** on the Create Performance Test dialog box to start recording your test case.

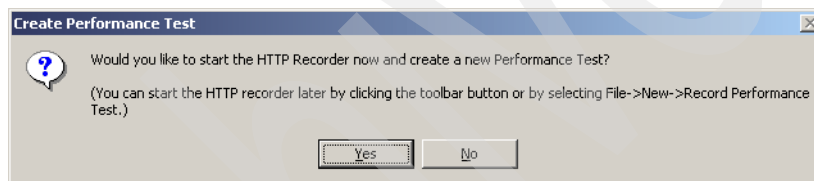


Figure 17-5 Create new performance test inside your project

4. Enter Trade6Scenario as the name for your recording in the HTTP Proxy Recorder dialog and click **Finish**.



Figure 17-6 Create new HTTP recording dialog

5. The HTTP Recorder will now start up and when it finishes initializing, it brings up an Internet Explorer window and displays an introduction page stating that you should not record confidential data.
6. Enter `http://http1.itso.ibm.com/trade/scenario` into the address field and press **Enter**. A page similar to the one in Figure 17-7 is displayed.

We have now finished recording and can start customizing the recorded script.

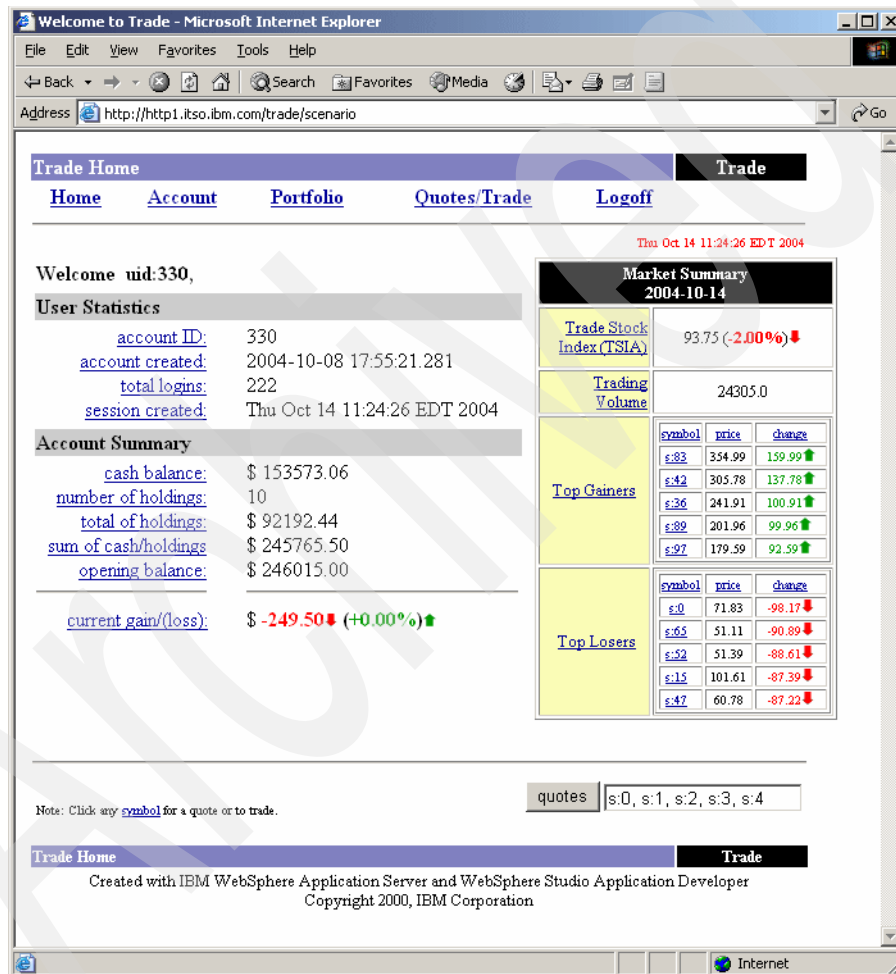


Figure 17-7 Internet Explorer recording

7. Close Internet Explorer. This displays the recorded script inside your workspace.

8. We now remove all references to images and cascading style sheets, since we only want to simulate load using the servlet provided by Trade 6 for this purpose. Right-click the resource(s) in the Test Contents pane of the Trade6Scenario view and click **Remove**.

Important: We are only removing the cascading style sheets and image files from this Performance Test because these are not relevant for our example. This is because the servlet that we are calling itself generates load and simulates a test scenario. This is a special case. If you want to simulate real user behavior, you usually do not want to perform this step!

9. Click **File** -> **Save** to save the changes made so far.

Creating a Performance Schedule

After setting up our project and recording a scenario, we can create a Performance Schedule. In a Performance Schedule, we basically assemble multiple scenarios with loops, delays, and other items to design a performance workload that matches the scenario we want to simulate as closely as possible.

1. Click **File** -> **New** -> **Performance Schedule** to create a new test schedule called Trade6Schedule in your workspace and click **Finish**. See Figure 17-8.

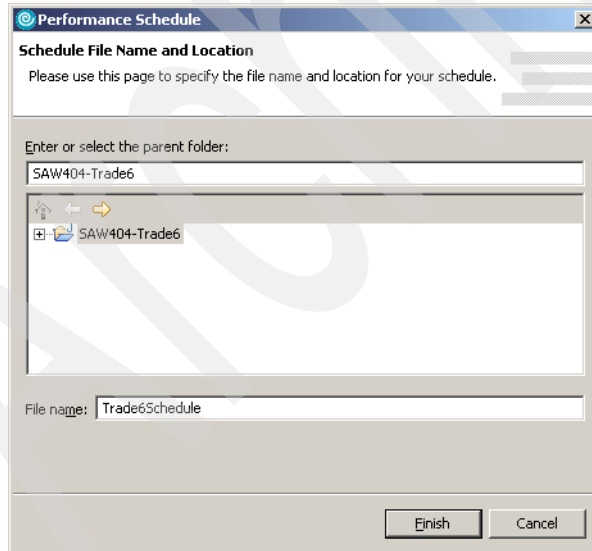


Figure 17-8 Create new Performance Schedule wizard

2. Right-click **User Group 1**, then click **Insert** -> **Loop**.

Note: Loops let you repeat tests and run a test at a rate that you specify.

3. Click **Loop** and set the Number of iterations to 50.
4. Right-click **Loop** -> **Insert** -> **Test**, select **Trade6Scenario** from the list and click **OK**.
5. Click **File** -> **Save** to save the changes.

Run a Performance Schedule

Now that we have everything recorded and set up, we can start to run our tests.

1. Right-click **Trade6Schedule**, then click **Run** -> **Performance Schedule**.
2. The workspace now displays a Performance Summary Report similar to the one in Figure 17-9 on page 963. This report is continuously updated during the test until all virtual test users have finished executing.

Important: As mentioned before, we have been using a Beta of the product when writing this redbook; the released product includes additional tabs in the report (in addition to Summary and Response versus Time). Please see the RPT 6.1 online help topic “Evaluating Results” for details on these additional tabs.

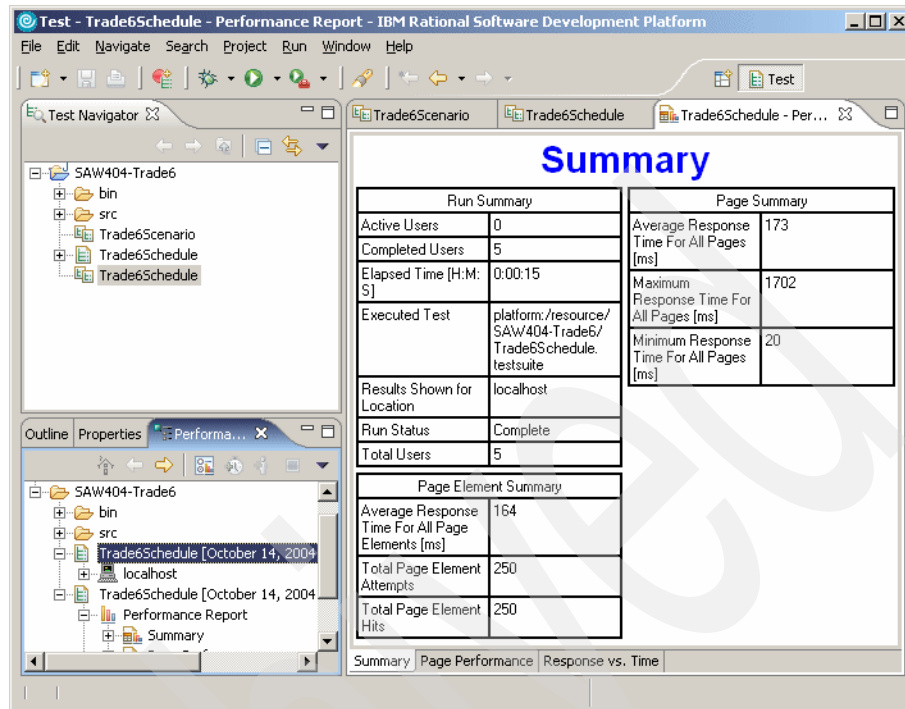


Figure 17-9 Performance Schedule summary report

One of the reports you can now look at to review the test results is the Response versus Time report shown in Figure 17-10 on page 964 that plots the average response time over time during your test execution.

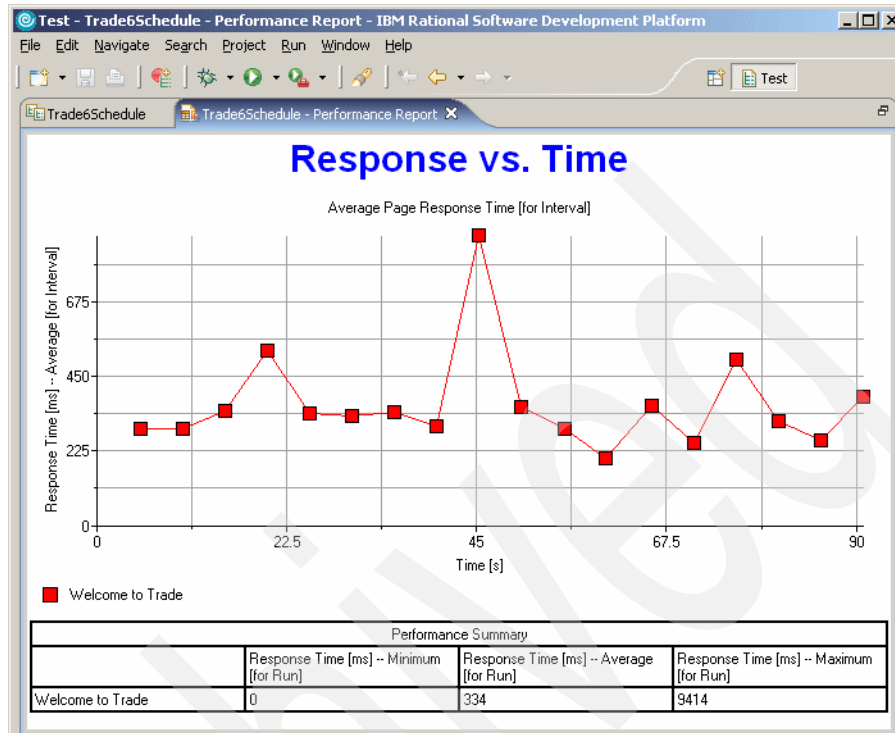


Figure 17-10 Rational Performance Tester Response versus Time graph

Please take the time now to look through the generated statistics and performance reports.

17.3.4 Other testing tools

There are a myriad of open-source and commercial products and services related to Web application testing. A search on Google for “Web application testing” can yield in excess of two million hits. Below is a list of just some of the commercial and free tools available. This list is by no means complete, and does not imply an endorsement or recommendation of the tools. It is merely provided as an alternative source of testing tools if ApacheBench, OpenSTA, or Rational Performance Tester are not used.

- **JMeter**, Open Source software available from the Apache Software Foundation at:

<http://jakarta.apache.org/jmeter/>

- **TestMaker** and **TestNetwork**, from PushToTest. See:

<http://www.pushtotest.com/>

- ▶ **Grinder.** See:
<http://grinder.sourceforge.net>
- ▶ **LoadRunner** from Mercury Interactive. See:
<http://www.mercury.com/us/products/performance-center/loadrunner/>
- ▶ **Segue SilkPerformer.** See:
<http://www.segue.com/products/load-stress-performance-testing/>
- ▶ **WebLOAD** from Radview. See:
<http://www.radview.com/products/WebLOAD.asp>
- ▶ **WebStone** from Mindcraft. See:
<http://www.mindcraft.com/webstone/>
- ▶ **OpenLoad** from Opendemand Systems. See:
<http://www.opendemand.com/openload/>

17.4 Performance monitoring guidelines

Before tuning measures can be taken, there is one important question that has to be answered first: “What part of the system shall I tune?”. Simple random tuning acts will seldom provide the desired effect, and sometimes will even produce worse results. This section discusses which components to examine first, and gives a practical hotlist of the top ten monitors and which tools to use. Finally, it makes a short excursion into performance analysis and load testing best practices.

17.4.1 Top ten monitoring hotlist

If you experience performance problems, start using Tivoli Performance Viewer and walk through the following top ten monitoring items checklist. It will help you to more easily find those significant parts of your system where the most performance impact can be gained by tuning. It cannot be stated often enough that performance tuning is not a science, but an art, so do not expect miracles by adjusting a few “knobs” inside WebSphere Application Server: every system and every application behaves differently, and requires different tuning measures! But the Performance Monitoring Infrastructure (PMI), Tivoli Performance Viewer and the Performance Advisors will assist you on the way to achieving your goal: a system configured for optimum performance.

Because of the sheer magnitude of monitors and tuning parameters, knowing where to start, what to monitor and which component to tune first is hard. Follow these top ten monitoring steps to check the most important counters and metrics

of WebSphere Application Server. See Figure 17-11 for a graphical overview of the most important resources to check. Consider the following:

- ▶ If you recognize something out of the ordinary, for example, an overutilized thread pool or a JVM that spends 50% in garbage collection at peak load time, then concentrate your tuning actions there first.
- ▶ Perform your examination when the system is under typical production level load and make note of the observed values.
- ▶ Alternatively, save Tivoli Performance Viewer sessions to the file system, where the monitoring data will be stored for recurring analysis.
- ▶ Keep in mind that one set of tuning parameters for one application will not work the same way for another.

For tips and good starting points on which direction to tune a specific parameter, refer to 17.5, “Performance tuning guidelines” on page 975 for respective tuning measures related to this checklist.

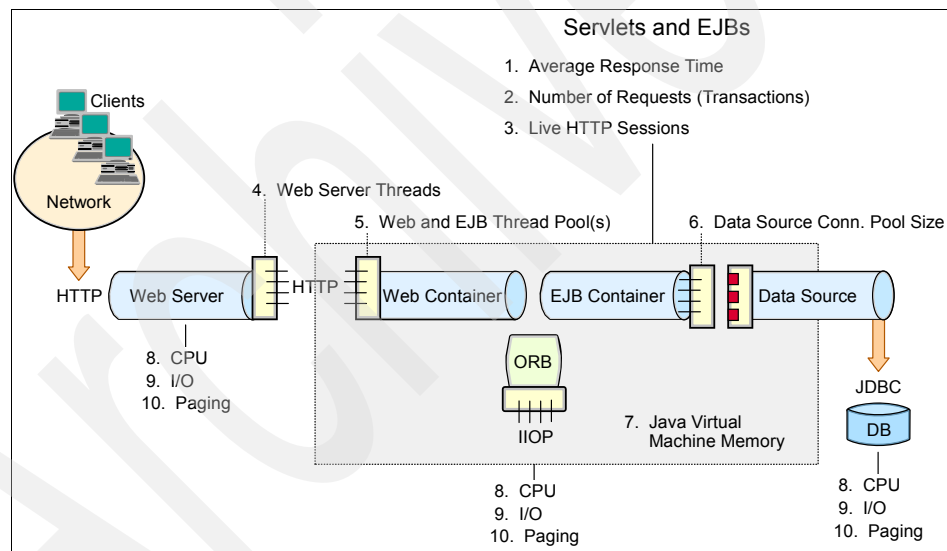


Figure 17-11 Top ten monitoring items checklist

Note: For details about the various monitoring levels available, please refer to 14.2.4, “PMI predefined statistic sets” on page 780.

Servlets and Enterprise JavaBeans

1. Average response time

To find this value in the Tivoli Performance Viewer, click **Performance Modules** -> **Web Applications** and look at the value named *ServiceTime*.

2. Number of requests (Transactions)

To find this value in the Tivoli Performance Viewer, click **Performance Modules** -> **Web Applications** and look at the value named *RequestCount*.

3. Live number of HTTP Sessions

To find this value in the Tivoli Performance Viewer, click **Performance Modules** -> **Servlet Session Manager** and look at the value named *Live Count*.

Thread pools

4. Web server threads

See 14.9, “Monitoring the IBM HTTP Server” on page 826 for details on how to enable this in IBM HTTP Server or Apache. For all other Web servers, please refer to the manual of your Web server product.

5. Web container and EJB container thread pool

All WebSphere Application Server thread pools can be viewed in the Thread Pools summary report described in 14.3.5, “Tivoli Performance Viewer summary reports” on page 796.

Data sources

6. Datasource connection pool size

All datasource connection pools can be viewed in the Connection Pools summary report described in 14.3.5, “Tivoli Performance Viewer summary reports” on page 796.

Java Virtual Machine

7. Java Virtual Machine (JVM) memory, garbage collection statistics

Information provided about the JVM runtime depends on the debug settings of the JVM and can be viewed in the Tivoli Performance Viewer in **Performance Modules** -> **JVM Runtime**. For more information, please refer to 14.2.6, “Using JVMPi facility for PMI statistics” on page 786.

System resources on Web, application, and Database servers

8. CPU utilization

9. Disk and network I/O

10. Paging activity

Tivoli Performance Advisor

Use the Tivoli Performance Advisor in addition to Tivoli Performance Viewer. Together, these tools will help you to tune your system, and the Tivoli Performance Advisor will give you recommendations on inefficient settings.

View recommendations and data from the Performance Advisor by clicking **Advisor** in the Tivoli Performance Viewer. For detailed instructions on how to use the Performance Advisor, refer to 14.7, “Performance Advisors” on page 813.

Hint: If TPA complains that the instrumentation level for the JVM is not sufficient, be aware that you might also need to activate the JVMPI facility. See 14.2.6, “Using JVMPI facility for PMI statistics” on page 786 for instructions on how to enable JVMPI logging.

Be aware that enabling JVMPI puts additional overhead on your system. Therefore, you should only enable it when needed and disable it after your tests.

Runtime Performance Advisor

The Runtime Performance Advisor service runs inside each application server process and has to be enabled explicitly. It uses the same rules engine as Tivoli Performance Advisor and will produce similar tuning recommendations. In contrast to Performance Advisor in Tivoli Performance Viewer, the Runtime Performance Advisor does not provide the full spectrum of advice types, but is suited for long-term analysis. For a detailed explanation of how to use the Runtime Performance Advisor, refer to 14.7, “Performance Advisors” on page 813.

Tip: It is important to configure the correct number of processors in the Runtime Performance Advisor configuration panel, otherwise the recommendations can be inaccurate.

However, keep in mind that both Tivoli Performance Advisor and Runtime Performance Advisor advice can only be accurate if your application runs with production level load, and without exceptions and errors. This is best done by running a performance test, where a production level load can be simulated in a reproducible manner. Both advisors need the CPU utilization to be high to provide good (and complete) advice, which is best accomplished during a load and stress test. If this is not the case, resulting output may be misleading and most certainly contradictory.

17.4.2 Performance analysis

Web performance analysis describes the process of finding out how well your system (a Web application or generally a Web site) performs, and pinpointing performance problems caused by inadequate resource utilization, such as memory leaks or over- or underutilized object pools, to name just a few. Once you know the trouble spots of your system, you can take counter-measures to reduce their impact by taking appropriate tuning actions.

Terminology

Performance analysis is a very comprehensive topic. It fills entire books and is surely out of the scope of this book. We attempt to provide a short introduction to that subject. Following is a concise definition of the three most important concepts used in performance analysis literature:

- ▶ Load
- ▶ Throughput
- ▶ Response Time

Load

A Web site, and especially the application that is running behind it, typically behaves and performs differently depending on the current load, that is, the number of users that are concurrently *using* the Web site at a given point in time. This includes clients who actively perform requests at a time, but also clients who are currently reading a previously requested Web page. Peak load often refers to the maximum number of concurrent users using the site at some point in time.

Response time

Response time refers to the timeframe from the time the client initiates a request until it receives the response. Typically, the time taken to display the response (usually the HTML data inside the browser window) is also accounted for in the response time.

Throughput

A Web site can only handle a specific number of concurrent requests. Throughput depends on that number and on the average time a request takes to process; it is measured in requests/second. If the site can handle 100 concurrent requests and the average response time is one second, the Web site's throughput is 100 requests per second.

Performance testing

To find out about the performance of your system in the first place, you have to run a performance test to get an idea of how many users will be able to access the site at the same time without noteworthy delays. Even at a later time, load tests are most helpful to find out how much performance was gained by a

specific tuning measure. After each tuning step, the load test has to be repeated (10-15 iterations of it are not uncommon), until the system meets your performance objectives. For performance assessment and tuning, the following test requirements can be stated:

- ▶ The load expected on the system has to be defined.

That implies that you have to create a model of your expected real-world, production level workload, based on your experience and knowledge of your clients' behavior. Using this representative model in combination with a stress test tool, you will create as many testcases as needed, and combine them into a testsuite that simulates the desired user behavior and performs the necessary interactions with your site during a load test. In some cases, when no good estimate of such a model can be given, even a crude approximation is better than performing no load test at all, and it can always be refined for future test cycles.

- ▶ The load test has to be repeatable.

Tuning without the ability to perform repeatable tests is very hard, because of the lack of comparable performance data. To be repeatable, it is necessary to restore the initial system state after each test iteration. Usually persistent application or transaction data is stored in databases or back-end systems, so that implies the following requirement:

A second, staging database and/or back-end system has to be used for testing purposes. Changes to the data on that system will not have consequences in the real world: meaning, the placing of a thousand orders in an online shop during load testing will not activate the order fulfillment process in the back end.

- ▶ Find the optimum load level.

The goal is to find the site's saturation point. First, the system has to be driven even over the point where performance begins to degrade. This is commonly referred to as *drawing the throughput curve* (see Figure 17-12 on page 971).

Start a series of tests to plot the throughput curve. Begin testing with wide open server queues (pools) to allow maximum concurrency in your system. Record the throughput (average requests per second) and the response time (average time for requests), increasing the concurrent user load after each test. You will find the system's saturation point in the diagram where the throughput becomes constant (the maximum throughput point is reached), but response time begins to grow proportionally with the user load.

Refer to 17.5.4, "Adjusting WebSphere Application Server system queues" on page 977 for more information about the queuing network and "Determining optimum queue sizes" on page 980 for detailed explanation of the steps on drawing the throughput curve.

Important: Here the goal is to drive CPU utilization to nearly 100 percent. If you cannot reach that state with opened up queues, there is most likely a bottleneck in your application, in your network connection, some hard limit on a resource or possibly any other external component you did not modify.

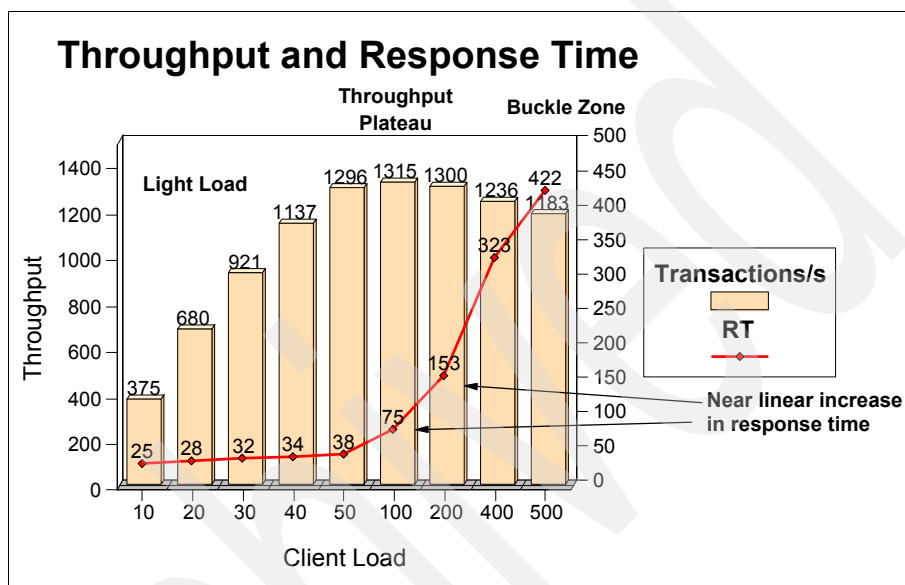


Figure 17-12 How to find the optimum load level for a system: the saturation point

The Monitoring - Tuning - Testing cycle

Perform an initial load test to get a *baseline* that you can refer to later, with the Performance Advisors facility enabled. Use the number of users at your saturation point as the load parameter in all future load tests. Check the monitors in Tivoli Performance Viewer according to the top ten hotlist and the Performance Advisors. After you have implemented an advisor's recommendation, perform another test cycle to find out if you have gained throughput (because of a decreased average response time) or freed resources (for example, memory by reducing thread pools) which you could then possibly spend on other components that need more memory resources. Repeat this procedure to find an optimum configuration for your system and application. Keep in mind that performance tuning is an iterative process, so do not rely on results from a single run.

Here is a short overview of the steps necessary to perform a load test for tuning purposes with the Runtime Performance Advisor as described above:

1. Enable the PMI service in all application servers and Node Agent (if using a WebSphere Network Deployment environment), and restart both. In WebSphere V6, PMI is enabled by default with the Basic monitoring set, so unless you disabled it previously, there should be no need to restart the servers. However, you might want to configure the statistics to be monitored for your test.
2. Enable Runtime Performance Advisor for all application servers.
3. Open **Monitoring and Tuning -> Performance Viewer -> Current Activity** and select the server you want to monitor in the WebSphere Administrative Console.
4. Simulate your representative production level load using a stress test tool.
 - Make sure that there are no errors or exceptions during the load test.
 - Record throughput and average response time statistics to plot a curve at the end of all testing iterations.
5. Check the Runtime Performance Advisor and Tivoli Performance Viewer. Apply advice and follow your own intuition. You'll get messages like the one in Figure 17-13 on page 973 that provide suggestions about which settings might improve performance on your system.
 - Restart components or services if necessary.
 - Reset all components (for example, the database) to the initial state.
6. Retest (go back to step 4).

Attention: Please notice that advice provided by Tivoli Performance Viewer Performance Advisor or the Runtime Performance Advisor is not applied automatically by WebSphere. You'll have to decide which changes make sense for your environment, apply them to your configuration, and test the effects these changes had during a load test.

Message

TUNE0214W: The session cache for Trade#tradeWeb.war is smaller than the average number of live sessions. Increasing the session cache size to at least 1200.0 may improve performance. Additional explanatory data follows. Session cache size (the maximum in memory session count): 1,000. Current live sessions: 1,300. Average live sessions over the last sampling interval: 1,200. This alert has been issued 1 time(s) in a row. The threshold will be updated to reduce the overhead of the analysis.

Message type

Warning

Explanation

In this situation, the overflow session cache is used instead of the main session cache, possibly hurting performance. Check that the session growth is bounded. In general, the average number of live sessions is approximately the rate of session creation times the average lifetime of a session.

User action

From the administrative console, click: Servers > Application Servers > Server > Web Container Settings > Web Container > Session Management.

Message Originator

com.ibm.ws.performance.tuning.serverAlert.TraceResponse

Source object type

RasLoggingService

Timestamp

Oct 11, 2004 5:09:27 PM EDT

Thread Id

c

Node name

app1Node

Server name

Web1

Figure 17-13 Tuning advice given by the Runtime Performance Advisor

Data capture best practices

To get accurate results, mind the following best practices for data capturing:

- Measure during steady-state of the load test

Do not include ramp-up/ramp-down times in your performance data measurements and analysis (see Figure 17-14 on page 974). Measure during the steady-state when the maximum of users are concurrently performing requests.

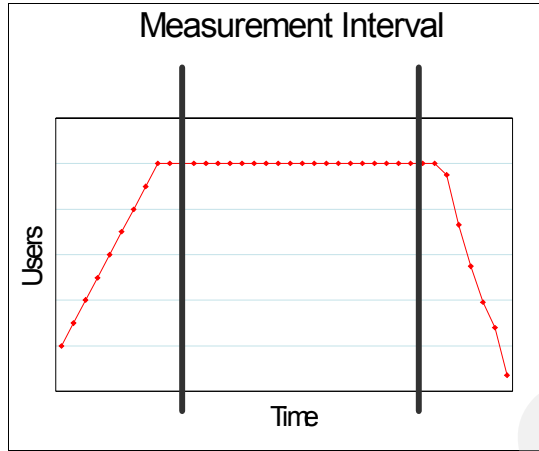


Figure 17-14 Measurement interval: concurrently active users versus elapsed time

- Monitor machine logs and resources

Monitor important log files for exceptions or errors. Be sure that there are no exceptions or deadlocks in the database. Keep an eye on system resources like memory, paging activity, CPU, disk IO, network utilization, socket states, etc., for bottlenecks.

Important log files are SystemOut.log and SystemErr.log. Monitor these logs to make sure your application runs without errors. SystemErr.log should typically remain free of entries. Errors logged there *must* be solved before you can capture meaningful performance data. Likewise, any sort of exception in SystemOut.log during the performance run should be solved before another run, because exception handling and the I/O necessary to write stacks to the log are expensive operations that impact performance.

For a Network Deployment cluster, you don't need to repeat all the monitoring steps for each cluster member if they are all set up identically; monitoring one or two representative cluster members should be sufficient. What is essential however, is to check the CPU statistics for each node to make sure that all cluster member processes are using similar amounts of CPU and there are no extra processes consuming CPU on any nodes that can interfere with the application server CPU efficiency.

- Test on idle systems

Make sure you do not perform test runs during database backups, maintenance cycles, or while other people perform tests on these systems.

- Use isolated networks

Load driving machines should be attached to the same network switch as your first-layer machines like Web servers or network dispatchers to be able

to apply the highest load possible on the front network interfaces of your infrastructure.

- Performance tuning is an iterative process
10-15 test runs are quite usual during the tuning phase. Perform long lasting runs to detect resource leaks, for example, memory leaks, where the load tested application runs out of heap space only after a given time.

17.5 Performance tuning guidelines

Tuning is about utilizing resources to their fullest potential, resulting in the fastest request processing possible. Many components in WebSphere Application Server have an impact on performance and tuning is highly application-dependent. This section discusses how to identify bottlenecks, gives a practical introduction into analyzing them, and finally gives recommendations on settings for major WebSphere environment properties.

It is important to stress once again that performance tuning is not an exact science. Factors that influence testing vary from application to application, and also from platform to platform. This section is designed to provide a primer for the reader in a way that describes areas that can be tuned to increase performance.

The latest performance tuning information can be found in the WebSphere Application Server V6 InfoCenter, located at

<http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp>

Select the appropriate version of the InfoCenter you wish to read. Then navigate to the **Tuning performance** section in the contents pane on the left side.

17.5.1 Tuning parameter hotlist

The following list of tuning suggestions is a subset of the complete list that follows in this chapter. These parameters are on a hot list because they have an important impact on performance. The majority of the parameters are application dependent, and should only be adjusted after close inspection of testing results for the application in question.

- Hardware and capacity settings, see 17.5.3, “Hardware and capacity settings” on page 976.
- Java Virtual Machine heap size, see 17.5.6, “Java tuning” on page 1002.
- Application assembly performance checklist, see 17.5.5, “Application assembly performance checklist” on page 991.

- ▶ Data sources connection pool and prepared statement cache, see “Data sources” on page 983.
- ▶ Solaris operating system TCP_TIME_WAIT_INTERVAL, see “Solaris” on page 1014.
- ▶ Pass by value versus Pass by reference, see “Pass by reference” on page 1022.
- ▶ IBM HTTP Server access logs, see “Access logs” on page 1019.
- ▶ HTTP keep-alive connections, see “HTTP transport channel Maximum persistent requests” on page 988.

17.5.2 Parameters to avoid failures

The following list of parameters is a subset of the entire list of parameters, and is designed to minimize application failures. The majority of these will also be application specific and should be adjusted based on observed application execution and configuration:

- ▶ Number of Connections to DB2:
Default settings are likely too low. See “Data sources” on page 983.
- ▶ Allow thread allocation beyond maximum:
When selected this might cause problems because the system becomes overloaded because too many threads are allocated. See “Thread pool” on page 988 for more information.
- ▶ Using TCP Sockets for DB2 on Linux:
For local databases. See “Use TCP sockets for DB2 on Linux” on page 1026.
- ▶ Connection Pool Size:
Ensure enough connections for transaction processing with Entity EJBs and for avoiding deadlock. See “Connection pool size” on page 983.

17.5.3 Hardware and capacity settings

Obviously, the hardware and its capacity play an important role in performance. Part 1 to Part 5 of this redbook provide detailed information about how to set up a scalable WebSphere environment. However, the following parameters are to be considered:

- ▶ Disk speed
Disk speed and configuration can have a dramatic effect on the performance of application servers that run applications that are heavily dependent on database support, that use extensive messaging, or are processing workflow.

Disk I/O subsystems that are optimized for performance, for example RAID array, are essential components for optimum application server performance in these environments. It is recommended that you spread the disk processing across as many disks as possible to avoid contention issues that typically occur with one or two disk systems.

- ▶ Processor speed

Increasing the processor speed often helps throughput and response times once other bottlenecks have been removed where the processor was waiting for such events as input and output, and application concurrency.

- ▶ System memory

Increasing memory to prevent the system from paging memory to disk improves the performance. Allow a minimum of 256 MB of memory for each processor (512 MB is recommended). Adjust the parameter when the system is paging and processor utilization is low because of the paging.

- ▶ Networks

Run network cards and network switches at full duplex. Running at half duplex decreases performance. Verify that the network speed can accommodate the required throughput. Also, make sure that 100 MB is in use on 10/100 Ethernet networks.

17.5.4 Adjusting WebSphere Application Server system queues

WebSphere Application Server establishes a queuing network, which is a group of interconnected queues that represent various components. There are queues established for the network, Web server, Web container, EJB container, Object Request Broker (ORB), data source, and possibly a connection manager to a custom back-end system. Each of these resources represents a queue of requests waiting to use that resource. Queues are load-dependent resources. As such, the average response time of a request depends on the number of concurrent clients.

As an example, think of an application, consisting of servlets and EJBs, that accesses a back-end database. Each of these application elements reside in the appropriate WebSphere component (for example servlets in the Web container) and each component can handle a certain number of requests in a given time frame.

A client request enters the Web server and travels through WebSphere components in order to provide a response to the client. Figure 17-15 on page 978 illustrates the processing path this application takes through the WebSphere components as interconnected pipes that form a large tube.

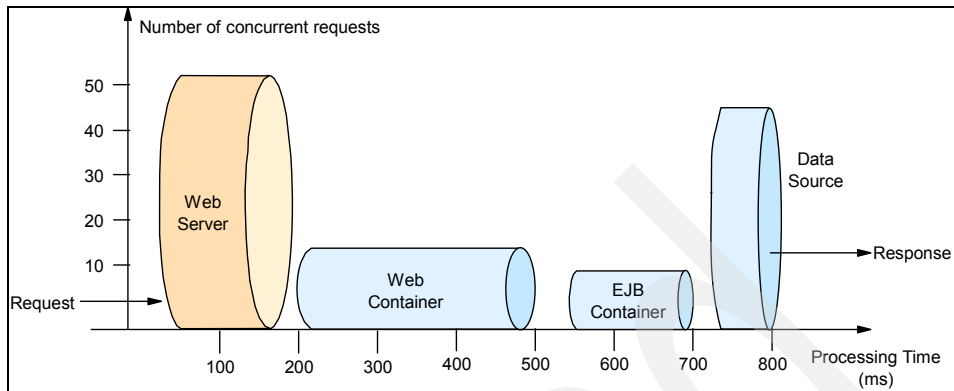


Figure 17-15 Queuing network

The width of the pipes (illustrated by height) represents the number of requests that can be processed at any given time. The length represents the processing time that it takes to provide a response to the request.

In order to find processing bottlenecks, it is useful to calculate a transactions per second (tps) ratio for each component. Ratio calculations for a fictional application are shown in Example 17-3:

Example 17-3 Transactions per second ratio calculations

The Web Server can process 50 requests in 100 ms = $\frac{50req}{0.1s} = 500tps$

The Web container parts can process 18 requests in 300 ms = $\frac{18req}{0.3s} = 60tps$

The EJB container parts can process 9 requests in 150 ms = $\frac{9req}{0.15s} = 60tps$

The datasource can process 40 requests in 50 ms = $\frac{40req}{0.05s} = 800tps$

The example shows that the application elements in the Web container and in the EJB container process requests at the same speed. Nothing would be gained from increasing the processing speed of the servlets and/or Web container because the EJB container would still only handle 60 transactions per second. The requests normally queued at the Web container would simply shift to the queue for the EJB container.

This example illustrates the importance of queues in the system. Looking at the operations ratio of the Web and EJB containers, each is able to process the same number of requests over time. However, the Web container could produce twice the number of requests that the EJB container could process at any given time. In order to keep the EJB container fully utilized, the other half of the requests must be queued.

It should be noted that it is common for applications to have more requests processed in the Web server and Web container than by EJBs and back-end systems. As a result, the queue sizes would be progressively smaller moving deeper into the WebSphere components. This is one of the reasons queue sizes should not solely depend on the operation ratios.

The following section outlines a methodology for configuring the WebSphere Application Server queues. The dynamics of an individual system can be dramatically changed by moving resources, for example moving the database server onto another machine, or providing more powerful resources, for example a faster set of CPUs with more memory. Thus, adjustments to the tuning parameters are for a specific configuration of the production environment.

Queuing before WebSphere

The first rule of tuning is to minimize the number of requests in WebSphere Application Server queues. In general, requests should wait in the network (in front of the Web server), rather than waiting in WebSphere Application Server. This configuration allows only those requests that are ready to be processed to enter the queuing network. To accomplish this, specify that the queues furthest upstream (closest to the client) are slightly larger, and that the queues further downstream (furthest from the client) are progressively smaller.

As an example, the queuing network becomes progressively smaller as work flows downstream. When 200 client requests arrive at the Web server, 125 requests remain queued in the network because the Web server is set to handle 75 concurrent clients. As the 75 requests pass from the Web server to the Web container, 25 remain queued in the Web server and the remaining 50 are handled by the Web container. This process progresses through the data source until 25 user requests arrive at the final destination, the database server. Because there is work waiting to enter a component at each point upstream, no component in this system must wait for work to arrive. The bulk of the requests wait in the network, outside of WebSphere Application Server. This type of configuration adds stability, because no component is overloaded. The Edge Server Components can be used to direct waiting users to other servers in a WebSphere Application Server cluster.

Note: If resources are more readily available on the application server or database server, it may be appropriate to tune such that every request from the Web server has an available application server thread, and every application server thread has an available database connection. The need for this type of configuration depends on the application and overall site design.

Determining optimum queue sizes

A simple way to determine the right queue size for any component is to perform a number of load runs against the application server environment at a time when the queues are very large, ensuring maximum concurrency through the system.

For example, one approach would be:

- ▶ Set the queue sizes for the Web server, Web container and data source to an initial value, for example 100.
- ▶ Simulate a large number of typical user interactions entered by concurrent users in an attempt to fully load the WebSphere environment. In this context, “concurrent users” mean simultaneously active users that send a request, wait for the response, and immediately resend a new request upon response reception - without thinktime.

Use any stress tool to simulate this workload, such as OpenSTA, discussed in “Testing the performance of an application” on page 940 or the tools mentioned in “Other testing tools” on page 964.

- ▶ Measure overall throughput and determine at what point the system capabilities are fully stressed (the saturation point).
- ▶ Repeat the process, each time increasing the user load. After each run, record the throughput (requests per second) and response times (seconds per request) and plot the throughput curve.

The throughput of WebSphere Application Server is a function of the number of concurrent requests present in the total system. At some load point, congestion will start to develop due to a bottleneck and throughput will increase at a much lower rate until reaching a saturation point (maximum throughput value). The throughput curve should help you identify this load point.

It is desirable to reach the saturation point by driving CPU utilization close to 100%, since this gives an indication that a bottleneck is not caused by something in the application. If the saturation point occurs before system utilization reaches 100%, there is likely another bottleneck that is being aggravated by the application. For example, the application might be creating Java objects causing excessive garbage collection mark phase bottlenecks in Java that you may notice as only one processor being utilized at a time on multi-processor systems. On uni-processor systems you'll not be able to see the symptom, but only the

problems it is causing. Please refer to section 17.5.6, “Java tuning” on page 1002 to learn more about this behavior.

Note: There are two ways to manage application bottlenecks: remove the bottleneck or replicate the bottleneck. The best way to manage a bottleneck is to remove it. You can use a Java-based application profiler, such as Rational Application Developer (see Chapter 15, “Development-side performance and analysis tools” on page 839 for more information), Performance Trace Data Visualizer (PTDV), Optimizelt, JProbe or Jinsight to examine overall object utilization.

The most manageable type of bottleneck occurs when the CPUs of the servers become fully utilized. This type of bottleneck can be fixed by adding additional or more powerful CPUs.

An example throughput curve is shown in Figure 17-16.

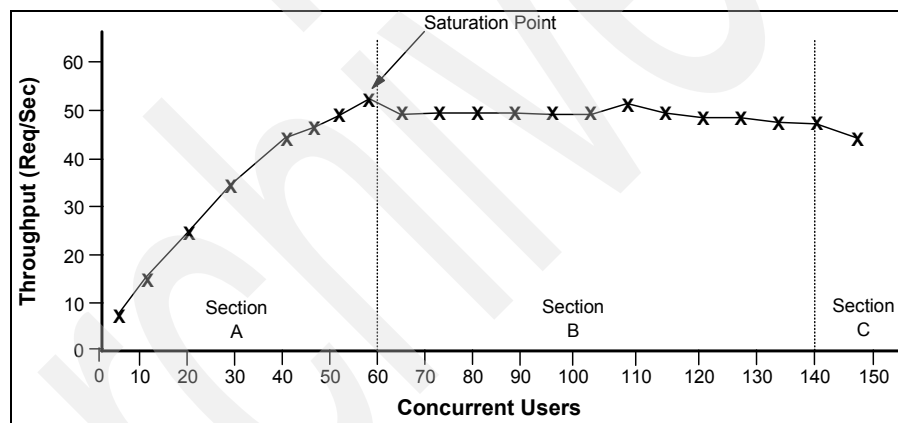


Figure 17-16 Throughput curve

In Figure 17-16, Section A contains a range of users that represent a light user load. The curve in this section illustrates that as the number of concurrent user requests increase, the throughput increases almost linearly with the number of requests. You can interpret this to mean that at light loads, concurrent requests face very little congestion within the WebSphere Application Server system queues.

In the heavy load zone or Section B, as the concurrent client load increases, throughput remains relatively constant. However, the response time increases proportionally to the user load. That is, if the user load is doubled in the heavy load zone, the response time doubles.

In Section C (the buckle zone) one or more of the system components have become exhausted and throughput starts to degrade. For example, the system might enter the buckle zone when the network connections at the Web server exhaust the limits of the network adapter or if the requests exceed operating system limits for file handles.

Determining the maximum concurrency point

The number of concurrent users at the throughput saturation point represents the maximum concurrency of the application. For example, if the application saturated the application server at 50 users, 48 users might give the best combination of throughput and response time.

This value is called the Max Application Concurrency value. Max Application Concurrency becomes the preferred value for adjusting the WebSphere Application Server system queues. Remember, it is desirable for most users to wait in the network; therefore, queue sizes should decrease when moving downstream farther from the client. For example, given a Max Application Concurrency value of 48, start with system queues at the following values: Web server 75, Web container 50, data source 45. Perform a set of additional experiments adjusting these values slightly higher and lower to find the best settings.

The Tivoli Performance Viewer (TPV) can be used to determine the number of concurrent users through the Servlet Engine Thread Pool Concurrently Active Threads metric. Refer to 14.3, “Using Tivoli Performance Viewer” on page 790 for more information about TPV.

Adjusting queue settings for access patterns

In many cases, only a fraction of the requests passing through one queue enters the next queue downstream. In a site with many static pages, many requests are fulfilled at the Web server and are not passed to the Web container. In this circumstance, the Web server queue can be significantly larger than the Web container queue. In the previous section, the Web server queue was set to 75 rather than closer to the value of Max Application Concurrency. Similar adjustments need to be made when different components have different execution times. As the percentage of static content decreases, however, a significant gap in the Web server queue and the application server queue can create poorly performing sites overall. Remember that tuning is an art, not a science, and different Web sites have different requirements.

For example, in an application that spends 90% of its time in a complex servlet and only 10% making a short JDBC query, on average 10% of the servlets are using database connections at any time, so the database connection queue can be significantly smaller than the Web container queue. Conversely, if much of a

servlet execution time is spent making a complex query to a database, consider increasing the queue values at both the Web container and the data source. Always monitor the CPU and memory utilization for both the WebSphere Application Server and the database servers to ensure the CPU or memory are not being overutilized.

Configuring the queues

Within WebSphere Application Server, the queues are represented as pooled resources, for example thread pools or database connection pools. Pool settings determine the maximum concurrency level of a resource. This section describes how the different queues are represented in WebSphere and their settings.

As the queues are most easily tuned from the inside out of the WebSphere Application Server environment, this section describes the queue properties in this order (looking at the components from right to left in Figure 17-15 on page 978).

Data sources

There are two settings to be concerned with for determining data source queues:

- ▶ Connection pool size
- ▶ Prepared statement cache size

Connection pool size

When accessing any database, the initial database connection is an expensive operation. WebSphere Application Server supports JDBC 2.0 Standard Extension APIs to provide support for connection pooling and connection reuse. The connection pool is used for direct JDBC calls within the application, as well as for enterprise beans using the database.

Tivoli Performance Viewer can help find the optimal size for the connection pool. Use a standard workload that represents a typical number of incoming client requests, use a fixed number of iterations, and use a standard set of configuration settings. Watch the Pool Size, Percent Used and Concurrent Waiters counters of the data source entry under the JDBC Connection Pools module. The optimal value for the pool size is that which reduces the values for these monitored counters. If Percent Used is consistently low, consider decreasing the number of connections in the pool.

Better performance is generally achieved if the value for the connection pool size is set lower than the value for the Max Connections in the Web container. Lower settings for the connection pool size (10-30 connections) typically perform better than higher (more than 100) settings. On UNIX platforms, a separate DB2 process is created for each connection. These processes quickly affect performance on systems with low memory, causing errors.

Each entity bean transaction requires an additional connection to the database specifically to handle the transaction. Be sure to take this into account when calculating the number of data source connections.

The connection pool size is set from the Administrative Console using these steps:

1. Select **Resources** -> **JDBC Providers** in the console navigation tree.
2. Select the appropriate scope (cell, node or server), depending on your configuration.
3. Open the JDBC provider configuration by clicking the name of the provider.
4. Select the **Data Sources** entry under Additional Properties.
5. Open the data source configuration by clicking the data source name.
6. Select **Connection pool properties**.
7. Use the Minimum connections and Maximum connections fields to configure the pool size.
8. Save the configuration and restart the affected application servers for the changes to take effect.

The default values are 1 for Min connections and 10 for Max connections.

Deadlock can occur if the application requires more than one concurrent connection per thread, and the database connection pool is not large enough for the number of threads. Suppose each of the application threads requires two concurrent database connections and the number of threads is equal to the maximum connection pool size. Deadlock can occur when both of the following are true:

- ▶ Each thread has its first database connection, and all are in use.
- ▶ Each thread is waiting for a second database connection, and none would become available, since all threads are blocked.

To prevent the deadlock in this case, the value set for the database connection pool must be at least one higher than the number of waiting threads in order to have at least one thread complete its second database connection.

To avoid deadlock, code the application to use, at most, one connection per thread. If the application is coded to require *C* concurrent database connections per thread, the connection pool must support at least the following number of connections, where *T* is the maximum number of threads:

$$T * (C - 1) + 1$$

The connection pool settings are directly related to the number of connections that the database server is configured to support. If the maximum number of connections in the pool is raised, and the corresponding settings in the database are not raised, the application fails and SQL exception errors are displayed in the SystemErr.log file.

Prepared statement cache size

The data source optimizes the processing of prepared statements to help make SQL statements process faster. It is important to configure the cache size of the data source to gain optimal statement execution efficiency. A prepared statement is a precompiled SQL statement that is stored in a prepared statement object. This object is used to efficiently execute the given SQL statement multiple times. If the JDBC driver specified in the data source supports precompilation, the creation of the prepared statement will send the statement to the database for precompilation. Some drivers might not support precompilation and the prepared statement might not be sent until the prepared statement is executed.

If the cache is not large enough, useful entries will be discarded to make room for new entries. In general, the more prepared statements your application has, the larger the cache should be. For example, if the application has five SQL statements, set the prepared statement cache size to 5, so that each connection has five statements.

Tivoli Performance Viewer can help tune this setting to minimize cache discards. Use a standard workload that represents a typical number of incoming client requests, use a fixed number of iterations, and use a standard set of configuration settings. Watch the PrepStmtCacheDiscardCount counter of the JDBC Connection Pools module. The optimal value for the statement cache size is the setting used to get either a value of zero or the lowest value for PrepStmtCacheDiscardCount.

As with the connection pool size, the statement cache size setting requires resources at the database server. Specifying too large a cache could have an impact on database server performance. It is highly recommended that you consult your database administrator for determining the best setting for the prepared statement cache size.

Note: The statement cache size setting defines the maximum number of prepared statements cached per connection.

The cache size is set from the Administrative Console using these steps:

1. Select **Resources** -> **JDBC Provider** in the console navigation tree.
2. Select the appropriate scope (cell, node or server), depending on your configuration.

3. Open the JDBC provider configuration by clicking the name of the provider.
4. Select the **Data Sources** entry in the Additional Properties pane.
5. Open the data source configuration by clicking the data source name.
6. Select **WebSphere Application Server data source properties**.
7. Use the Statement cache size field to configure the total cache size.
8. Save the configuration and restart the affected application servers for the change to take effect.

EJB container

You can use the following parameters to make adjustments that improve performance for the EJB container.

Cache settings (Cache size and Cleanup interval)

To determine the cache absolute limit, multiply the number of enterprise beans active in any given transaction by the total number of concurrent transactions expected. Then, add the number of active session bean instances. Use the Tivoli Performance Viewer to view bean performance information. The cache settings consist of two parameters: the cache size and the cleanup interval.

The cleanup interval specifies the interval at which the container attempts to remove unused items from the cache in order to reduce the total number of items to the value of the cache size.

The cache size specifies the number of buckets in the active instance list within the EJB container.

To change these settings, click **Servers -> Application servers -> <AppServer_Name> -> EJB Container Settings -> EJB container -> EJB cache settings**.

The default values are Cache size=2053 buckets and Cache cleanup interval=3000 milliseconds.

ORB thread pool size

Method invocations to enterprise beans are only queued for requests coming from remote clients going through the RMI activity service. An example of such a client is an EJB client running in a separate Java Virtual Machine (another address space) from the enterprise bean. In contrast, no queuing occurs if the EJB client (either a servlet or another enterprise bean) is installed in the same JVM that the EJB method runs on and the same thread of execution as the EJB client.

Remote enterprise beans communicate by using the RMI/IIOP protocol. Method invocations initiated over RMI/IIOP are processed by a server-side ORB. The thread pool acts as a queue for incoming requests. However, if a remote method request is issued and there are no more available threads in the thread pool, a new thread is created. After the method request completes, the thread is destroyed. Therefore, when the ORB is used to process remote method requests, the EJB container is an open queue, due to the use of unbounded threads.

Tivoli Performance Viewer can help tune the ORB thread pool size settings. Use a standard workload that represents a typical number of incoming client requests, use a fixed number of iterations, and use a standard set of configuration settings. Watch the PercentMaxed counter of the Thread Pools module. If the value of this counter is consistently in the double digits, then the ORB could be a bottleneck and the number of threads in the pool should be increased.

The degree to which the ORB thread pool value needs to be increased is a function of the number of simultaneous servlets (that is, clients) calling enterprise beans and the duration of each method call. If the method calls are longer or the applications spend a lot of time in the ORB, consider making the ORB thread pool size equal to the Web container size. If the servlet makes only short-lived or quick calls to the ORB, servlets can potentially reuse the same ORB thread. In this case, the ORB thread pool can be small, perhaps even one-half of the thread pool size setting of the Web container.

The ORB thread pool size is configured from the Administrative Console using these steps:

To change these settings, click ->**EJB cache settings**.

1. Select **Servers -> Application servers -> <AppServer_Name> -> Container Services**.
2. Select **ORB Service -> Thread Pool**.
3. Use the Maximum Size field to configure the maximum pool size. Note that this only affects the number of threads held in the pool (the actual number of ORB threads can be higher).
4. Save the configuration and restart the affected application server for the change to take effect.

Some additional settings related to the ORB can also be tuned. These are explained in 17.5.12, "Object Request Broker (ORB)" on page 1022.

Web container

To route servlet requests from the Web server to the Web containers, a transport connection between the Web server plug-in and each Web container is established. The Web container manages these connections through transport channels and assigns each request to a thread from the Web container thread pool.

Thread pool

The Web container maintains a thread pool to process inbound requests for resources in the container (that is servlets and JSPs).

Tivoli Performance Viewer can help tune the Web container thread pool size settings. Use a standard workload that represents a typical number of incoming client requests, use a fixed number of iterations, and use a standard set of configuration settings. Watch the `PercentMaxed` and `ActiveCount` counters of the Thread Pools module. If the value of the `PercentMaxed` counter is consistently in the double digits, then the Web container could be a bottleneck and the number of threads should be increased. On the other hand if the number of active threads are significantly lower than the number of threads in the pool, consider lowering the thread pool size for a performance gain.

The Web container thread pool size is configured from the Administrative Console using these steps:

1. Select **Servers -> Application servers -> <AppServer_Name>**.
2. Select the **Thread Pools** entry under Additional Properties.
3. Select the **WebContainer** entry in the thread pools list of the workspace.
4. Use the Maximum Size field to configure the maximum pool size. Note that in contrast to the ORB, the Web container only uses threads from the pool, hence a closed queue. The default value is 50.
5. Save the configuration and restart the affected application server for the change to take effect.

Important: Checking the **Allow thread allocation beyond maximum thread size** box on the Thread Pool Configuration page allows for an automatic increase of the number of threads beyond the maximum size configured for the thread pool. As a result of this, the system can become overloaded because too many threads are allocated.

HTTP transport channel Maximum persistent requests

The maximum persistent requests is the maximum number of requests allowed on a single keep-alive connection. This parameter can help prevent denial of service attacks when a client tries to hold on to a keep-alive connection. The

Web server plug-in keeps connections open to the application server as long as it can, providing optimum performance.

A good starting value for the maximum number of requests allowed is 100 (which is the default value). If the application server requests are received from the Web server plug-in only, increase this parameter's value.

The maximum number of requests allowed is configured from the Administrative Console using these steps:

1. Select **Servers -> Application servers -> <AppServer_Name>**.
2. Select **Web Container Settings -> Web container transport chains** under Container Settings.
3. Select the transport chain you want to modify, for example **WCInboundDefault**.
4. Select **HTTP Inbound Channel (HTTP #)** in the Transport Channels pane.
5. Enter a value in the Maximum persistent requests field.
6. Click **OK**.
7. Save the configuration and restart the affected application server for the change to take effect.

HTTP transport channel Read timeout

Specifies the amount of time, in seconds, the HTTP transport channel waits for a read request to complete on a socket after the first read request occurs. The read being waited for could be an HTTP body (such as a POST) or part of the headers if they were not all read as part of the first read request on the socket.

The read timeout is configured from the Administrative Console using these steps:

1. Select **Servers -> Application servers -> <AppServer_Name>**.
2. Select **Web Container Settings -> Web container transport chains** under Container Settings.
3. Select the transport chain you want to modify.
4. Select **HTTP Inbound Channel (HTTP #)** in the Transport Channels pane.
5. Enter a value in the Read timeout field (the default starting value is 60).
6. Click **OK**.
7. Save the configuration and restart the affected application server for the change to take effect.

Web server

The queue mechanism regarding the Web server in WebSphere Application Server V6 was greatly improved in its scalability and tuning can focus primarily on the Web server itself. You do not have to take into account about what load this places onto the application servers. All Web servers have settings for configuring the maximum number of concurrent requests accepted. For information about how to configure this setting, refer to 17.5.8, “The Web server” on page 1015. For details about the new features of IBM HTTP Server powered by Apache 2.0 and how to use them, refer to “IBM HTTP Server powered by Apache 2.0” on page 1015.

Determining the Web server maximum concurrency threads setting

A Web server monitor (see 14.9, “Monitoring the IBM HTTP Server” on page 826) can help tune the maximum concurrent thread processing setting for the Web server. Use a standard workload that represents a typical number of incoming client requests, use a fixed number of iterations, and use a standard set of configuration settings. Watch the number of Web server threads going to the Web container and the number of threads accessing static content locally on the Web server.

Using cluster configurations

The capability to spread workload among application servers using clustering can be a valuable asset in configuring highly scalable production environments. This is especially true when the application is experiencing bottlenecks that are preventing full CPU utilization of Symmetric Multiprocessing (SMP) servers. When adjusting the WebSphere system queues in clustered configurations, remember that when a server is added to a cluster, the server downstream receives twice the load. This is illustrated in Figure 17-17.

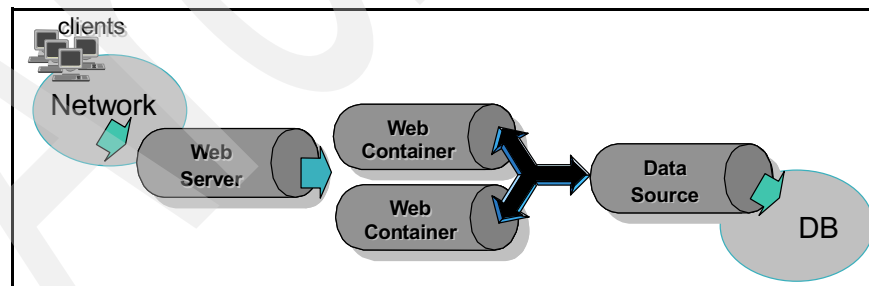


Figure 17-17 Clustering and queuing

Note: While the database exists once in the environment and is configured once in a cell, each application server that uses a datasource has its own datasource loaded in its own JNDI namespace.

Two Web containers within a cluster are located between a Web server and a data source. It is assumed the Web server, Web container, and data source (but not the database) are all running on a single SMP server. Given these constraints, the following queue considerations need to be made:

- ▶ Web server queue settings can be doubled to ensure ample work is distributed to each Web container.
- ▶ Web container thread pools can be reduced to avoid saturating a system resource such as CPU or another resource that the servlets are using.
- ▶ The data source pools can be reduced to avoid saturating the database server.
- ▶ Java heap parameters can be reduced for each instance of the application server. For versions of the JVM shipped with WebSphere Application Server, it is crucial that the heap from all JVMs remain in physical memory. Therefore, if a cluster of four JVMs are running on a system, enough physical memory must be available for all four heaps.

17.5.5 Application assembly performance checklist

Application assembly tools are used to assemble J2EE components and modules into J2EE applications. Generally, this consists of defining application components and their attributes including enterprise beans, servlets and resource references. Many of these application configuration settings and attributes play an important role in the runtime performance of the deployed application. The most important parameters and advice for finding optimal settings are:

- ▶ Enterprise bean modules:
 - Entity EJBs - Bean cache
 - Method extensions - Isolation level
 - Method extensions - Access intent
 - Container transactions
- ▶ Web modules:
 - Web application - Distributable
 - Web application - Reload interval
 - Web application - Reload enabled
 - Web application - Web components - Load on startup

Enterprise bean modules

This section explains the enterprise bean module parameters mentioned above in detail.

Note: Although WebSphere Application Server 5.1 and higher also supports EJB 2.0, the following information refers to EJB 1.1 settings.

Entity EJBs - Bean cache

WebSphere Application Server provides significant flexibility in the management of database data with Entity EJBs. The **Entity EJBs Activate at** and **Load at** configuration settings specify how and when to load and cache data from the corresponding database row data of an enterprise bean. These configuration settings provide the capability to specify enterprise bean caching Options A, B or C, as specified in the EJB 1.1 specification.

Option A provides maximum enterprise bean performance by caching database data outside of the transaction scope. Generally, Option A is only applicable where the EJB container has exclusive access to the given database. Otherwise, data integrity is compromised. Option B provides more aggressive caching of Entity EJB object instances, which can result in improved performance over Option C, but also results in greater memory usage. Option C is the most common real-world configuration for Entity EJBs.

► Bean cache - Activate at

This setting specifies the point at which an enterprise bean is activated and placed in the cache. Removal from the cache and passivation are also governed by this setting. Valid values are **Once** and **Transaction**. **Once** indicates that the bean is activated when it is first accessed in the server process, and passivated (and removed from the cache) at the discretion of the container, for example when the cache becomes full. **Transaction** indicates that the bean is activated at the start of a transaction and passivated (and removed from the cache) at the end of the transaction. The default value is **Transaction**.

► Bean cache - Load at

This setting specifies when the bean loads its state from the database. The value of this property implies whether the container has exclusive or shared access to the database. Valid values are **Activation** and **Transaction**. **Activation** indicates the bean is loaded when it is activated and implies that the container has exclusive access to the database. **Transaction** indicates that the bean is loaded at the start of a transaction and implies that the container has shared access to the database. The default is **Transaction**.

The settings of the Activate at and Load at properties govern which commit options are used.

- For Option A (exclusive database access), use Activate at = Once and Load at = Activation.

This option reduces database input/output by avoiding calls to the `ejbLoad` function, but serializes all transactions accessing the bean instance. Option A can increase memory usage by maintaining more objects in the cache, but can provide better response time if bean instances are not generally accessed concurrently by multiple transactions.

Note: When using WebSphere Network Deployment and workload management is enabled, Option A cannot be used. You must use settings that result in the use of Options B or C.

- For Option B (shared database access), use Activate at = Once and Load at = Transaction.

Option B can increase memory usage by maintaining more objects in the cache. However, because each transaction creates its own copy of an object, there can be multiple copies of an instance in memory at any given time (one per transaction), requiring the database be accessed at each transaction. If an enterprise bean contains a significant number of calls to the `ejbActivate` function, using Option B can be beneficial because the required object is already in the cache. Otherwise, this option does not provide significant benefit over Option A.

- For Option C (shared database access), use Activate at = Transaction and Load at = Transaction.

This option can reduce memory usage by maintaining fewer objects in the cache. However, there can be multiple copies of an instance in memory at any given time (one per transaction). This option can reduce transaction contention for enterprise bean instances that are accessed concurrently but not updated.

Method extensions - Isolation level

WebSphere Application Server enterprise bean method extensions provide settings to specify the level of transactional isolation used when accessing data.

Isolation level settings specify various degrees of runtime data integrity provided by the corresponding database. First, choose a setting that meets data integrity requirements for the given application and specific database characteristics.

The valid values are:

- Serializable

- ▶ Repeatable read
- ▶ Read committed
- ▶ Read uncommitted

Isolation level also plays an important role in performance. Higher isolation levels reduce performance by increasing row locking and database overhead while reducing data access concurrency. Various databases provide different behavior with respect to the isolation settings. In general, Repeatable read is an appropriate setting for DB2 databases. Read committed is generally used for Oracle. Oracle does not support Repeatable read and will translate this setting to the highest isolation level of Serializable.

The Isolation level can be specified at the bean or method level. Therefore, it is possible to configure different isolation level settings for various methods. This is an advantage when some methods require higher isolation than others, and can be used to achieve maximum performance while maintaining integrity requirements. However, isolation cannot change between method calls within a single enterprise bean transaction. A runtime exception will be thrown in this case.

The following section describes the four isolation levels:

- ▶ Serializable

This level prohibits the following types of reads:

- *Dirty reads*: A transaction reads a database row containing uncommitted changes from a second transaction.
- *Nonrepeatable reads*: One transaction reads a row, a second transaction changes the same row, and the first transaction rereads the row and gets a different value.
- *Phantom reads*: One transaction reads all rows that satisfy an SQL WHERE condition, a second transaction inserts a row that also satisfies the WHERE condition, and the first transaction applies the same WHERE condition and gets the row inserted by the second transaction.

- ▶ Repeatable read

This level prohibits dirty reads and nonrepeatable reads, but it allows phantom reads.

- ▶ Read committed

This level prohibits dirty reads, but allows nonrepeatable reads and phantom reads.

- ▶ Read uncommitted

This level allows dirty reads, nonrepeatable reads, and phantom reads.

The container uses the transaction isolation level attribute as follows:

- ▶ Session beans and entity beans with bean-managed persistence (BMP):
For each database connection used by the bean, the container sets the transaction isolation level at the start of each transaction unless the bean explicitly sets the isolation level on the connection.
- ▶ Entity beans with container-managed persistence (CMP):
The container generates database access code that implements the specified isolation level.

Isolation levels for EJBs

WebSphere Application Server enterprise bean method extensions provide settings to specify individual enterprise bean methods as read-only. This setting denotes whether the method can update entity attribute data (or invoke other methods that can update data in the same transaction).

By default, all enterprise bean methods are assumed to be "update" methods. This results in EJB Entity data always being persisted back to the database at the close of the enterprise bean transaction. Marking enterprise methods that do not update entity attributes as Access Intent Read, provides a significant performance improvement by allowing the WebSphere Application Server EJB container to skip the unnecessary database update.

A behavior for "finder" methods for CMP Entity EJBs is available. By default, WebSphere Application Server will invoke a Select for Update query for CMP enterprise bean finder methods such as `findByPrimaryKey`. This exclusively locks the database row for the duration of the enterprise bean transaction. However, if the enterprise bean finder method has been marked as Access Intent Read, the container will not issue the For Update on the select, resulting in only a read lock on the database row.

Important: The following applies only to EJBs that are compliant with the EJB 1.1 specification. We recommend that new EJBs be developed to be compliant with the EJB 2.0 specification. However, some existing EJBs may only be compliant with EJB 1.1.

Isolation levels and read-only methods: If an entity bean has methods that do not update attributes (getter type methods) they can be specified as read-only methods in the deployment description. This will avoid executing a SELECT FOR UPDATE/UPDATE pair of SQL statements, and hence will provide a performance improvement.

EJBs deployed in WebSphere Application Server can be assigned one of four transaction isolation levels. These are Repeatable Read, Read Committed, Read

Uncommitted and Serializable. The default level is Repeatable Read, but performance can be improved by the use of a lower isolation level such as Read Committed.

Important: The following applies only to EJBs that are compliant with the EJB 2.0 specification.

EJB 2.0 has adopted the J2C resources isolation level. Now you can specify the isolation by using the access intent in your EJBs. The access intent for each method of an EJB 2.0 CMP entity bean can be specified in the EJB deployment descriptor to provide hints to the EJB container about how the method will be used. The supported access intents are pessimistic update - Weakest Lock At Load, pessimistic update, optimistic update, pessimistic read, and optimistic read. In general, read-only methods and optimistic locking will provide better performance. However, be careful with the use of optimistic locking since problems may only become apparent under heavy load and hence may not be found in development.

Table 17-1 describes the access intent settings, and how this might affect the underlying isolation levels.

Table 17-1 Access intent settings

Access intent profile name	Concurrency control, access type	Transaction isolation
wsPessimisticRead	Pessimistic/Read (lock held for duration of transaction)	Repeatable Read
wsPessimisticUpdate	Pessimistic/Update (generates SELECT FOR UPDATE and grabs locks at beginning of transaction)	Repeatable Read
wsPessimisticUpdate-Exclusive	Pessimistic/Update (lock held for duration of transaction)	Serializable
wsPessimisticUpdate-noCollision	Pessimistic/Update (no locks are held, updates permitted, locks escalated on update)	Read Committed
wsPessimisticUpdate-WeakestLockAtLoad (default)	Pessimistic/Update (no locks are held, updates permitted, locks escalated on update)	Repeatable read
wsOptimisticRead	Optimistic/Read (read-only access)	Read Committed

Access intent profile name	Concurrency control, access type	Transaction isolation
wsOptimisticUpdate	Optimistic/Update (generates overqualified SQL UPDATE statements and forces compare before update)	Read Committed

You must balance your performance needs against the data consistency. As stated before, with more restrictive isolation levels, the performance becomes worse. So you have to study your application to see when to use one access intent setting or another. Some core entity beans can represent data involved in many transactions. If they do not have a fast access they will become a bottleneck. But also as the data is used in many transactions the data must be consistent. So you can apply different isolation levels to different methods.

The get methods are read-only methods, so you can apply a low isolation level as Read Committed. The set methods normally are more seldom used and need to be very strongly isolated, so you can apply the most restrictive isolation level as Serializable.

Pessimistic methods block the data for more time than optimistic methods, so normally optimistic methods are faster than pessimistic methods.

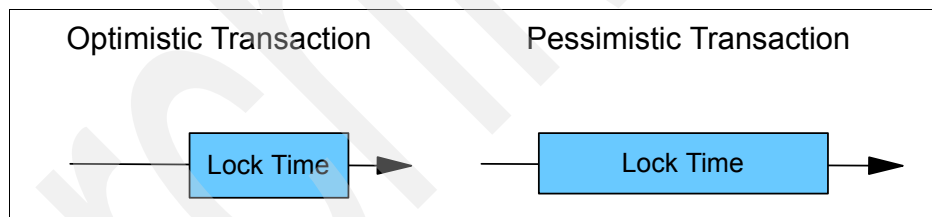


Figure 17-18 Lock time in Optimistic and Pessimistic transactions

In WebSphere Application Server V6 the access intent settings can be adjusted during runtime, depending on the requirements of the actual unit of work performed. This feature is part of Application profiling. Application profiling enables you to configure multiple access intent policies on the same entity bean. Application profiling reflects the fact that different units of work have different use patterns for enlisted entities and can require different kinds of support from the server run time environment.

Application Profiling

Application profiling enables you to identify particular units of work to the WebSphere Application Server run time environment. The run time can tailor its support to the exact requirements of that unit of work. Access intent is currently

the only run time component that makes use of the application profiling functionality. For example, you can configure one transaction to load an entity bean with strong update locks and configure another transaction to load the same entity bean without locks.

Application profiling introduces two new concepts in order to achieve this function: tasks and profiles.

► Tasks

A task is a configurable name for a unit of work. Unit of work in this case means either a transaction or an ActivitySession. The task is used for the duration of its unit of work to identify configured policies specific to that unit of work.

► Profiles

A profile is simply a mapping of a task to a set of access intent policies that are configured on entity beans. When an invocation on a bean (whether by a finder method, a CMR getter, or a dynamic query) requires data to be retrieved from the back end system, the current task associated with the request is used to determine the exact requirement of the transaction. The same bean loads and behaves differently in the context of the task-to-profile mapping. Each profile provides the developer an opportunity to reconfigure the application's access intent. If a request is operating in the absence of a task, the run time environment uses either a method-level access intent (if any) or a bean-level default access intent.

The Application Server Toolkit (AST) includes a static analysis engine that can assist you in configuring application profiling. The tool examines the compiled classes and the deployment descriptor of a Java 2 Platform, Enterprise Edition (J2EE) application to determine the entry point of transactions, calculate the set of entities enlisted in each transaction, and determine whether the entities are read or updated during the course of each identified transaction. You can execute the analysis in either closed world or open world mode. A closed-world analysis assumes that all possible clients of the application are included in the analysis and that the resulting analysis is complete and correct. The results of a closed-world analysis report the set of all transactions that can be invoked by a Web, JMS, or application client. The results exclude many potential transactions that never execute at run time. An open-world analysis assumes that not all clients are available for analysis or that the analysis cannot return complete or accurate results. An open-world analysis returns the complete set of possible transactions. The results of an analysis persist as an application profiling configuration. The tool establishes container managed tasks for servlets, JavaServer Pages (JSP) files, application clients, and message-driven beans (MDBs). Application profiles for the tasks are constructed with the appropriate access intent for the entities enlisted in the transaction represented by the task.

However, in practice, there are many situations where the tool returns at best incomplete results. Not all applications are amenable to static analysis. Some factory and command patterns make it impossible to determine the call graphs. The tool does not support the analysis of ActivitySessions. You should examine the results of the analysis very carefully. In many cases you must manually modify them to meet the requirements of the application. However, the tool can be an effective starting place for most applications and may offer a complete and quick configuration of application profiles for some applications.

Read-ahead hints

Read-ahead schemas enable applications to minimize the number of database round-trips by retrieving a working set of CMP beans for the transaction within one query. Read-ahead involves activating the requested CMP beans and caching the data of related beans (relationships), which ensures that data is present for the beans that are most likely to be needed next by an application. A read-ahead hint is a canonical representation of the related beans that are to be read. It is associated with a finder method for the requested bean type, which must be an EJB 2.x- compliant CMP entity bean.

Container transactions

The container transaction setting specifies how the container manages transaction scopes when delegating invocation to the enterprise bean individual business method. The legal values are:

- ▶ Never
- ▶ Mandatory
- ▶ Requires New
- ▶ Required
- ▶ Supports
- ▶ Not Supported
- ▶ Bean Managed

The container transactions attribute can be specified individually for one or more enterprise bean methods. Enterprise bean methods not requiring transactional behavior can be configured as Supports to reduce container transaction management overhead.

The following section describes the legal values in detail:

- ▶ Never

This legal value directs the container to invoke bean methods without a transaction context. If the client invokes a bean method from within a transaction context, the container throws the `java.rmi.RemoteException` exception.

If the client invokes a bean method from outside a transaction context, the container behaves in the same way as if the Not Supported transaction attribute was set. The client must call the method without a transaction context.

► **Mandatory**

This legal value directs the container to always invoke the bean method within the transaction context associated with the client. If the client attempts to invoke the bean method without a transaction context, the container throws the `javax.transaction.TransactionRequiredException` exception to the client. The transaction context is passed to any enterprise bean object or resource accessed by an enterprise bean method.

Enterprise bean clients that access these entity beans must do so within an existing transaction. For other enterprise beans, the enterprise bean or bean method must implement the Bean Managed value or use the Required or Requires New value. For non-enterprise bean EJB clients, the client must invoke a transaction by using the `javax.transaction.UserTransaction` interface.

► **Requires New**

This legal value directs the container to always invoke the bean method within a new transaction context, regardless of whether the client invokes the method within or outside a transaction context. The transaction context is passed to any enterprise bean objects or resources that are used by this bean method.

► **Required**

This legal value directs the container to invoke the bean method within a transaction context. If a client invokes a bean method from within a transaction context, the container invokes the bean method within the client transaction context. If a client invokes a bean method outside a transaction context, the container creates a new transaction context and invokes the bean method from within that context. The transaction context is passed to any enterprise bean objects or resources that are used by this bean method.

► **Supports**

This legal value directs the container to invoke the bean method within a transaction context if the client invokes the bean method within a transaction. If the client invokes the bean method without a transaction context, the container invokes the bean method without a transaction context. The transaction context is passed to any enterprise bean objects or resources that are used by this bean method.

► **Not Supported**

This legal value directs the container to invoke bean methods without a transaction context. If a client invokes a bean method from within a

transaction context, the container suspends the association between the transaction and the current thread before invoking the method on the enterprise bean instance. The container then resumes the suspended association when the method invocation returns. The suspended transaction context is not passed to any enterprise bean objects or resources that are used by this bean method.

► **Bean Managed**

This value notifies the container that the bean class directly handles transaction demarcation. This property can be specified only for session beans and (in EJB 2.0 implementations only) for message-driven beans, not for individual bean methods.

Web modules

This section explains the parameters that can be set for Web modules.

Web application - Distributable

The distributable flag for J2EE Web applications specifies that the Web application is programmed to be deployed in a distributed servlet container.

Important: Web applications should be marked as Distributable if, and only if, they will be deployed in a WebSphere Application Server clustered environment.

Web application - Reload interval

The reload interval specifies a time interval, in seconds, in which the Web application's file system is scanned for updated files, such as servlet class files or JSPs.

The Reload interval can be defined at different levels for various application components. Generally, the reload interval specifies the time the application server will wait between checks to see if dependent files have been updated and need to be reloaded. Checking file system time stamps is an expensive operation and should be reduced. The default is 0 (zero). Setting this to a value of 3 seconds is good for a test environment, because the Web site can be updated without restarting the application server. In production environments, checking a few times a day is a more common setting.

Web application - Reloading enabled

This specifies whether file reloading is enabled. The default is false.

Web application - Web components - Load on startup

Indicates whether a servlet is to be loaded at the startup of the Web application. The default is false.

Many servlets perform resource allocation and other up-front processing in the servlet `init()` method. These initialization routines can be costly at runtime. By specifying `Load on startup` for these servlets, processing takes place when the application server is started. This avoids runtime delays, which can be encountered on a servlet's initial access.

17.5.6 Java tuning

The following section focuses on tuning Java memory. Enterprise applications written in Java involve complex object relationships and utilize large numbers of objects. Although Java automatically manages memory associated with an object's *life cycle*, understanding the application's usage patterns for objects is important. In particular, ensure the following:

- ▶ The application is not over-utilizing objects
- ▶ The application is not leaking objects (that is, memory)
- ▶ The Java heap parameters are set to handle the use of objects

Understanding the effect of garbage collection is necessary to apply these management techniques.

Garbage collection basics

Garbage collection algorithms, like JVMs, have evolved and become more and more complex to understand. Knowledge of how the Garbage Collector (GC) works is necessary for designing and tuning Java applications and application servers. Following is a broad, somewhat simplified, overview of the Mark - Sweep - Compact (MSC) garbage collection technique implemented by IBM JVMs. For an in-depth study of additional, state-of-the-art heap management and garbage collection techniques, refer to the articles mentioned in “Additional JVM and garbage collection related resources” on page 1010.

The JVM allocates areas of storage inside the Java heap, where objects, arrays, and classes are stored. An allocated object is considered *live* when there exists at least one reference to it, that means, it is used by someone, commonly another object. Thus the object is also considered *reachable*. When this object is no longer used by anyone, all references should have been removed, it is now considered *garbage*, and its allocated storage area should be reclaimed for reuse. This task is performed by the Garbage Collector.

When the JVM is unable to allocate an object from the current Java heap because of lack of free, contiguous space, a memory allocation fault occurs (allocation failure) and the Garbage Collector is invoked. (The GC can also be invoked by a specific function call: `System.gc()`. However, when `System.gc()` is invoked, the JVM can simply 'take it under advisement' and choose to defer the GC operation until later if the JVM has more pressing needs to attend to.) The

first task of the GC is to make sure to acquire all locks required for garbage collection, and then stops all the other threads. Because of this garbage collection is also referred to as *stop-the-world* (STW) collection. Garbage collection will then take place in three phases: mark, sweep, and optionally compact.

Mark phase

In the mark phase, all *reachable* objects that are referenced either directly by the JVM, for example through threads stacks, or in turn by other objects, will be identified. Everything else that is not marked is considered garbage.

Sweep phase

All allocated objects that are not marked are swept away, that is, the space used by them is reclaimed.

Compaction phase

When the garbage has been removed from the heap, the GC can consider compacting the heap, which is typically riddled with holes caused by the freed objects by now. When there is no chunk of memory available big enough to satisfy an allocation request after garbage collection, the heap has to be compacted. Because heap compaction means moving objects around and updating all references to them, it is extremely costly in terms of time, and the GC tries to avoid it if possible. Modern JVM implementations try to avoid heap compaction by focusing on optimizing object placement in the heap.

Note: Do not confuse the Java heap with the native (or system) heap! The native heap is never garbage collected; it is used by the JVM process and stores all the objects that the JVM itself needs during its entire lifetime. The native heap is typically much smaller than the Java heap.

Heap expansion and shrinkage

Heap expansion will occur after garbage collection if the ratio of free to total heap size falls below the value specified by the `-Xminf` parameter. The default is 0.3 (or 30%).

Heap shrinkage will occur after garbage collection if the ratio of free to total heap size exceeds the value specified by the `-Xmaxf` parameter. The default is 0.6 (or 60%). The amount of expansion is governed by the minimum expansion size, set by the `-Xmine` parameter, and the maximum expansion size, defined by `-Xmaxe`. The defaults for `-Xmine` are 1MB, for `-Xmaxe` 0, which is equal to unlimited. These parameters do not have any effect on a fixed-size heap, where the `-Xms` and `-Xmx` values are equal.

Parallel versus concurrent operation

Recent releases of JVMs implement multiple helper threads that run in parallel on a multi-processor machine during the mark and sweep phases. These threads are asleep during normal operation, and only during garbage collection the work is divided between the main GC thread and his helper threads, using all processors simultaneously. *Parallel mark* mode is enabled by default since IBM JVM versions 1.3.0, while *parallel sweep* is enabled by default since version 1.3.1. As mentioned before, garbage collection is basically a stop-the-world operation. This is not entirely true: IBM JVM 1.3.1 and higher also know an optional *concurrent mark* mode, where a background thread is started by the JVM, and some of the work of the mark phase is done concurrently while all application threads are active. Thus the STW pause will be reduced when garbage collection finally occurs. Concurrent mark is disabled by default, and can be activated using the `-Xgcpolicy:optavgpause` parameter.

Note: In some cases, concurrent mark may reduce the throughput of an application. It is recommended you compare the application performance with and without concurrent mark, using identical loads to measure the effect on application performance.

In addition, IBM JVM 1.4 knows an *incremental compaction mode* which parallelizes the compaction phase. For an introductory explanation of parallel and concurrent modes, refer to the developerWorks article *Fine-tuning Java garbage collection performance* by Sumit Chawla found at:

<http://www.ibm.com/developerworks/ibm/library/i-gctroub/>

or the IBM JVM Diagnostics Guides (see “Additional JVM and garbage collection related resources” on page 1010) for an in-depth discussion of these features.

The garbage collection bottleneck

Examining Java garbage collection can give insight into how the application is utilizing memory. Garbage collection is a Java strength. By taking the burden of memory management away from the application developer, Java applications are more robust than applications written in languages that do not provide garbage collection. This robustness applies as long as the application is not abusing objects. Garbage collection normally consumes anywhere from 5 to 20% of the total execution time of a properly functioning application. If not managed, garbage collection can be one of the biggest bottlenecks for an application, especially when running on SMP server machines.

The garbage collection gauge

Garbage collection can be used to evaluate application performance health. By monitoring garbage collection during the execution of a fixed workload, users

gain insight as to whether the application is over-utilizing objects. Garbage collection can even be used to detect the presence of memory leaks.

Use the garbage collection and heap statistics in Tivoli Performance Viewer (see 14.3, “Using Tivoli Performance Viewer” on page 790) to evaluate application performance health. Mind that you have to enable the JVMPI facility (see “Performance data provided by JVMPI” on page 787) to get detailed garbage collection statistics. By monitoring garbage collection, memory leaks and overly used objects can be detected.

For this type of investigation, set the minimum and maximum heap sizes to the same value. Choose a representative, repetitive workload that matches production usage as closely as possible, user errors included. To ensure meaningful statistics, run the fixed workload until the state of the application is steady. Reaching this state usually takes several minutes.

Detecting over-utilization of objects

To see if the application is overusing objects, look in Tivoli Performance Viewer at the counters for the JVMPI profiler. The average time between garbage collection calls should be 5 to 6 times the average duration of a single garbage collection. If not, the application is spending more than 15% of its time in garbage collection. Also, look at the numbers of freed, allocated, and moved objects.

If the information indicates a garbage collection bottleneck, there are two ways to clear the bottleneck. The most cost-effective way to optimize the application is to implement object caches and pools. Use a Java profiler to determine which objects to target. If the application cannot be optimized, adding memory, processors and application clusters might help. Additional memory allows each application server in a cluster to maintain a reasonable heap size. Additional processors allow the cluster members to run in parallel.

Detecting memory leaks

Memory leaks in Java are a dangerous contributor to garbage collection bottlenecks. Memory leaks are more damaging than memory overuse, because a memory leak ultimately leads to system instability. Over time, garbage collection occurs more frequently until finally the heap is exhausted and Java fails with a fatal Out of Memory exception. Memory leaks occur when an unneeded object has references that are never deleted. This most commonly occurs in collection classes, such as Hashtable, because the table itself always has a reference to the object, even after real references have been deleted.

High workload often causes many applications to crash immediately after being deployed in the production environment. This situation is especially true for leaking applications where the high workload accelerates the magnification of the leakage and a memory allocation failure occurs.

Memory leak testing relates to magnifying numbers. Memory leaks are measured in terms of the amount of bytes or kilobytes that cannot be garbage collected. The delicate task is to differentiate these amounts from the expected sizes of useful and unusable memory. This task is achieved more easily if the numbers are magnified, resulting in larger gaps and easy identification of inconsistencies. The following is a list of important conclusions about memory leaks:

- ▶ Long-running test

Memory leak problems are manifested only after a period of time. Therefore, memory leaks are usually found during long-running tests. Short runs can lead to false alarms. One of the problems in Java is whether to say that a memory leak is occurring when memory usage has seemingly increased either abruptly or monotonically in a given period. These kind of increases can be valid, and the objects created can be referenced at a much later time. In other words, what method is used to differentiate the delayed use of objects from completely unused objects? Running applications over a long period of time will get a higher confidence for whether the delayed use of objects is actually occurring. Because of this, memory leak testing cannot be integrated with some other types of tests, such as functional testing, that occur earlier in the process. However, tests such as stress or durability tests can be integrated.

- ▶ System test

Some memory leak problems occur only when different components of a big project are combined and executed. Interfaces between components can produce known or unknown side effects. System test is a good opportunity to make these conditions happen.

- ▶ Repetitive test

In many cases, memory leak problems occur by successive repetitions of the same test case. The goal of memory leak testing is to establish a big gap between unusable memory and used memory in terms of their relative sizes. By repeating the same scenario over and over again, the gap is multiplied in a very progressive way. This testing helps if the amount of leaks caused by an execution of a test case is so minimal that it could hardly be noticed in one run.

Repetitive tests can be used at the system level or module level. The advantage with modular testing is better control. When a module is designed to keep the private module without creating external side effects such as memory usage, testing for memory leaks can be much easier.

- ▶ Concurrency test

Some memory leak problems can occur only when there are several threads running in the application. Unfortunately, synchronization points are very

susceptible to producing memory leaks because of the added complication in the program logic. Careless programming can lead to references being kept or unreleased. The incident of memory leaks is often facilitated or accelerated by increased concurrency in the system. The most common way to increase concurrency is to increase the number of clients in the test driver.

Consider the following when choosing which test cases to use for memory leak testing:

- ▶ A good test case exercises areas of the application where objects are created. Most of the time, knowledge of the application is required. A description of the scenario can suggest creation of data spaces, such as adding a new record, creating an HTTP session, performing a transaction and searching a record.
- ▶ Look at areas where collections of objects are being used. Typically, memory leaks are composed of objects of the same class. Also, collection classes such as Vector and Hashtable are common places where references to objects are implicitly stored by calling corresponding insertion methods. For example, the get method of a Hashtable object does not remove its reference to the object being retrieved.

Using Tivoli Performance Viewer to detect memory leaks

Tivoli Performance Viewer helps to find memory leaks. For best results, repeat experiments with increasing duration, such as 1000, 2000, and 4000-page requests. The Tivoli Performance Viewer graph of used memory should have a sawtooth shape. Each drop on the graph corresponds to a garbage collection. There is a memory leak if one of the following occurs:

- ▶ The amount of memory used immediately after each garbage collection increases significantly. The sawtooth pattern will look more like a staircase.
- ▶ The sawtooth pattern has an irregular shape.

Also, look at the difference between the number of objects allocated and the number of objects freed. If the gap between the two increases over time, there is a memory leak.

If heap consumption indicates a possible leak during a heavy workload (the application server is consistently near 100% CPU utilization), yet the heap appears to recover during a subsequent lighter or near-idle workload, this is an indication of heap fragmentation. Heap fragmentation can occur when the JVM is able to free sufficient objects to satisfy memory allocation requests during garbage collection cycles, but the JVM does not have the time to compact small free memory areas in the heap into larger contiguous spaces.

Another form of heap fragmentation occurs when small objects (less than 512 bytes) are freed. The objects are freed, but the storage is not recovered, resulting in memory fragmentation.

Java heap parameters

The Java heap parameters also influence the behavior of garbage collection. Increasing the heap size allows more objects to be created. Because a large heap takes longer to fill, the application runs longer before a garbage collection occurs. However, a larger heap also takes longer to compact and causes garbage collection to take longer.

For performance analysis, the initial and maximum heap sizes should be equal, as this will eliminate heap growing and shrinking delays. Equating initial with maximum heap size without previous heap size tuning will in most cases create an inefficiently used heap: when it is sized too big, the heap is not used by the application entirely and thus memory resources are wasted.

Important: It is not recommended that you set the initial and maximum heap sizes equal in a production environment. For details, please refer to the following technote:

<http://www.ibm.com/support/docview.wss?rs=180&uid=swg21160795>

When tuning a production system where the working set size of the Java application is not understood, a good starting value is to let the initial heap size be 25% of the maximum heap size. The JVM will then try to adapt the size of the heap to the working set size of the application.

Run a series of test experiments that vary the Java heap settings. For example, run experiments with 128 MB, 192 MB, 256 MB, and 320 MB. During each experiment, monitor the total memory usage. If the heap is expanded too aggressively, paging can occur (use the `vmstat` command or the Windows NT or Windows 2000 Performance Monitor to check for paging). If paging occurs, reduce the size of the heap or add more memory to the system.

Important: Make sure that the heap never pages, as that would introduce a enormous performance loss.

When all test runs are finished, compare the following statistics:

- ▶ Number of garbage collection calls
- ▶ Average duration of a single garbage collection call
- ▶ Average time between calls

- Ratio between the average length of a single garbage collection call and the average time between calls

If the application is not over-utilizing objects and has no memory leaks, a state of steady memory utilization is reached. Garbage collection also occurs less frequently and for shorter durations.

If the heap free time settles at 85% or more, consider decreasing the maximum heap size values, because the application server and the application are under-utilizing the memory allocated for heap.

The best result for the average time between garbage collections is at least five to six times the average duration of a single garbage collection. If you do not achieve this number, the JVM is spending more than 15% of its time in garbage collection. Finding this point might require trial and error. A longer interval between garbage collection cycles can result in longer GC runs, while very short intervals can be inefficient.

Heap thrashing

Avoid heap thrashing at all costs. It is caused by a heap that is barely large enough to avoid expansion but not large enough to satisfy future allocation failures. Usually a garbage collection cycle frees up enough space for not only the current allocation failure but a substantial number of future allocation requests. But when heap is getting thrashed, each garbage collection cycle frees up barely enough heap space to satisfy just the current allocation failure. The result is that the next allocation request leads to another garbage collection cycle, and so on. This scenario can also occur due to lots of short-lived objects.

Tuning the IBM JVM

Every JVM generally offers a whole set of tuning parameters affecting the performance of application servers and applications. Since JVM tuning is a wide and complex topic, this section will present an introduction into garbage collection analysis and tuning in reference to the IBM JVM and provide a series of links and resources for further study.

Analyzing verbosegc output

Using `verbosegc` as the next step after using Tivoli Performance Viewer, is a very good way to see what is going on with garbage collection. `Verbosegc` mode is enabled by the `-verbosegc` command-line option, and output is directed to the file `native_stderr.log`. This information can then be used to tune the heap size or diagnose problems. For excellent resources and illustrated examples of `verbosegc` analysis, refer to the following resources:

- ▶ *Sensible Sanitation: Understanding the IBM Java Garbage Collection Part 3: verbosegc and command-line parameters*

<http://www.ibm.com/developerworks/library/i-garbage3.html>

- ▶ *Fine-tuning Java garbage collection performance*

<http://www.ibm.com/developerworks/ibm/library/i-gctroub/>

Common problems

Following is a short list of common problems and suggested solutions:

- ▶ The GC frequency is too high until the heap reaches a steady state.
Use verbosegc to determine the size of the heap at a steady state and set -Xms to this value.
- ▶ The heap is fully expanded and the occupancy level is greater than 70%.
Increase the -Xmx value so the heap is not more than 70% occupied.
- ▶ At 70% occupancy the frequency of GCs is too great.
Change the setting of -Xminf. The default is 0.3, which will try to maintain 30% free space by expanding the heap. A setting of 0.4 increases this free space target to 40%, reducing the frequency of GCs.
- ▶ Pause times are too long.
Try using -Xgcpolicy:optavgpause (introduced in 1.3.1), which reduces pause times and makes them more consistent as the heap occupancy rises. There is a cost to pay in throughput. This cost varies and will be about 5%.

Additional JVM and garbage collection related resources

- ▶ *Garbage collection in the 1.4.1 JVM* available from developerWorks at
<http://www.ibm.com/developerworks/java/library/j-jtp11253/>
- ▶ *A brief history of garbage collection*, developerWorks
<http://www.ibm.com/developerworks/java/library/j-jtp10283/>
- ▶ *Sensible Sanitation: Understanding the IBM Java Garbage Collection, Parts 1 and 2*, developerWorks
<http://www.ibm.com/developerworks/ibm/library/i-garbage1/>
<http://www.ibm.com/developerworks/ibm/library/i-garbage2/>
- ▶ *IBM JVM Diagnostics Guides* from developerWorks
<http://www.ibm.com/developerworks/java/jdk/diagnosis/>
- ▶ *Fine-tuning Java garbage collection performance* from developerWorks
<http://www.ibm.com/developerworks/ibm/library/i-gctroub/>

- *Mash that trash - Incremental compaction in the IBM JDK Garbage Collector*

<http://www.ibm.com/developerworks/ibm/library/i-incrcomp/>

Tuning the Sun JVM

The Sun JVM also offers several tuning parameters affecting the performance of WebSphere Application Servers and application performance. These are explained in detail in the InfoCenter article “Java virtual machine settings”. The tuning settings can be set as Generic JVM Arguments. Therefore, look for the Generic JVM Arguments section in this article, especially for the parameters “-XX” and “-server | -client”.

The InfoCenter is available at:

<http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp>

Miscellaneous JVM settings

Following are several generic JVM setting recommendations that apply to most or all existing JVMs.

Just In Time (JIT) compiler

The Just In Time (JIT) compiler can significantly affect performance. If you disable the JIT compiler, throughput decreases noticeably. Therefore, for performance reasons, keep JIT enabled.

To determine the setting of this parameter:

8. Select **Servers -> Application servers -> <AppServer_Name>**.
9. Select **Java and Process Management -> Process Definition**.
10. Select **Java Virtual Machine**.
11. Check the setting of the Disable JIT check box.
12. If changes are made, save them and restart the application server.

Heap size settings

These parameters can set the maximum and initial heap sizes for the JVM.

In general, increasing the size of the Java heap improves throughput until the heap no longer resides in physical memory. After the heap begins swapping to disk, Java performance drastically suffers. Therefore, the maximum heap size needs to be low enough to contain the heap within physical memory.

The physical memory usage must be shared between the JVM and other applications running on the system, such as the database. For assurance, use a smaller heap, for example 64 MB, on machines with less memory.

Try a maximum heap of 128 MB on a smaller machine, that is, less than 1 GB of physical memory. Use 256 MB for systems with 2 GB memory, and 512 MB for larger systems. The starting point depends on the application.

If performance runs are being conducted and highly repeatable results are needed, set the initial and maximum sizes to the same value. This setting eliminates any heap growth during the run. For production systems where the working set size of the Java applications is not well understood, an initial setting of one-fourth the maximum setting is a good starting value. The JVM will then try to adapt the size of the heap to the working set of the Java application.

- ▶ Select **Servers -> Application servers -> <AppServer_Name>**.
- ▶ Select **Java and Process Management -> Process Definition**.
- ▶ Select **Java Virtual Machine**.
- ▶ In the General Properties configuration pane, enter values for the Initial Heap Size and Maximum Heap Size fields.
- ▶ Apply and save your changes, then restart the application server.

Class garbage collection

Disabling class garbage collection enables more class reuse, which, in some cases, has resulted in small performance improvements.

In most cases, run with class garbage collection turned on. This is the default.

To disable class garbage collection, enter the value `-Xnoclassgc` in the Generic JVM Arguments field of the application servers' JVM configuration. To do this:

- ▶ Select **Servers -> Application servers -> <AppServer_Name>**.
- ▶ Select **Java and Process Management -> Process Definition**.
- ▶ Select **Java Virtual Machine**.
- ▶ Enter the value `-Xnoclassgc` in the Generic JVM Arguments field.
- ▶ Apply and save your changes. Restart the application server.

17.5.7 Operating system tuning

This section provides some basic information about operating system tuning parameters, especially for AIX. For more details refer to the WebSphere InfoCenter at:

<http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp>

Expand **Tuning performance -> Tuning the application server environment**, then select **Tuning operating systems** or search for "Tuning operating systems".

Over time, more information will be added to the InfoCenter, for example HP-UX related tuning information. Therefore, for the latest information, always check the InfoCenter in addition to this redbook.

AIX

There are many AIX operating system settings to consider that are not within the scope of this redbook. Some of the settings you can adjust are:

- ▶ Adapter transmit and receive queue
- ▶ TCP/IP socket buffer
- ▶ IP protocol mbuf pool performance
- ▶ Update file descriptors
- ▶ Update the scheduler

However, two important settings are outlined in the next sections.

AIX with DB2

Separating your DB2 log files from the physical database files can boost performance. You can also separate the logging and the database files from the drive containing the Journaled File System (JFS) service. AIX uses specific volume groups and file systems for the JFS logging.

The AIX `filemon` utility is used to view all file system input and output, and to strategically select the file system for the DB2 logs. The default location for the files is `/home/<db2_instance>/<db2_instance>/NODExx/SQLyy/SQLLOGDIR/`.

To change the location of the files, at a DB2 command prompt, issue the following command:

```
db2 update db cfg for [database_name] using newlogpath  
[fully_qualified_path]
```

It is recommended that you move the logs to a separate disk when your application shows more than a 20% I/O wait time.

AIX file descriptors (ulimit)

Specifies the number of open files permitted.

When should you try adjusting this value? The default setting is typically sufficient for most applications. If the value set for this parameter is too low, a Memory allocation error is displayed.

Check the UNIX reference pages on `ulimit` for the syntax for different shells. For the KornShell shell (`ksh`), to set `ulimit` to 2000, issue the following command:

```
ulimit -n 2000
```

Use **smit** (or **smitty**) to permanently set this value for a user.

Use the command **ulimit -a** to display the current values for all limitations on system resources. The default setting is 2000, which is also the recommended value.

HP-UX 11i

Some HP-UX 11i settings can be modified to significantly improve WebSphere Application Server performance. For a number of HP-UX 11i kernel parameter recommendations see the InfoCenter article “Tuning HP-UX systems”.

Linux - RedHat Advanced Server 2.1

Kernel updates for RedHat Advanced Server 2.1 have implemented changes affecting WebSphere performance, especially memory-to-memory HTTP Session replication. If you are running any kernel prior to 2.4.9-e.23, upgrade at least to this kernel, but preferably to the latest supported.

Linux - SuSE Linux Enterprise Server 8 SP2A

The Linux scheduler is very sensitive to excessive context switching, so fixes have been integrated into the SLES8 kernel distribution to introduce delay when a thread yields processing. This fix is automatically enabled in SLES8 SP3 but must be enabled explicitly in SLES8 SP2A.

See the InfoCenter article “Tuning Linux systems” for details.

Solaris

There are several parameters that have a significant performance impact for WebSphere on Solaris. Please refer to the InfoCenter article “Tuning Solaris systems” for detailed information.

Also, many TCP parameters exist that can affect performance in a Solaris environment. For more information about tuning the TCP/IP Stack, see the article “Tuning your TCP/IP Stack and More” at:

<http://www.sean.de/Solaris/soltune.html>

Windows NT or 2000 TCP/IP parameters

Several tuning parameters exist for Windows NT and Windows 2000, such as TcpTimedWaitDelay and MaxUserPort.

Please see the InfoCenter article called “Tuning Windows systems” for more information.

17.5.8 The Web server

WebSphere Application Server provides plug-ins for several Web server brands and versions. Each Web server operating system combination has specific tuning parameters that affect the application performance.

This section discusses some of the performance tuning settings associated with the Web servers. In addition to the settings mentioned in this section, additional information about Web server tuning can be found in the WebSphere InfoCenter article called “Tuning Web servers”.

Web server configuration refresh interval

WebSphere Application Server administration tracks a variety of configuration information about WebSphere Application Server resources. Some of this information, such as URIs pointing to WebSphere Application Server resources, needs to be understood by the Web server. This configuration data is pushed to the Web server through the WebSphere Application Server plug-in at intervals specified by this parameter. Periodic updates allow new servlet definitions to be added without having to restart any of the WebSphere Application Server servers. However, the dynamic regeneration of this configuration information is costly in terms of performance. You can try to adjust this value in a stable production environment. The default reload interval setting is 60 seconds.

This parameter can be customized in the Administrative Console by selecting **Servers -> Web servers -> <WebServer_name> -> Plug-in properties** and setting the Refresh configuration interval there. Increase the refresh interval to a value that represents an acceptable wait time between the servlet update and the Web server update.

IBM HTTP Server powered by Apache 2.0

First we give you a short introduction into IBM HTTP Server powered by Apache 2.0 and its new features. For IHS 6.0 tuning information go to “Tuning IBM HTTP Server V6” on page 1017.

For the remainder of this section we’ll refer to the underlying product that IBM HTTP Server V2 and V6 are built on as Apache 2.0. Please notice that IBM HTTP Server incorporates all the features of Apache 2.0 and extends upon them, although we do not cover the specific features of IBM HTTP Server here.

New Features

There are various changes in IBM HTTP Server powered by Apache 2.0 and while it was an option in previous versions, WebSphere Application Server V6 is the first version to ship with this version by default. Following are a few important new features of IBM HTTP Server powered by Apache 2.0. For a complete feature list, visit the Apache 2.0 Web site at

http://httpd.apache.org/docs-2.0/new_features_2_0.html

► Apache 2.0 is a thread-based Web server

Although Apache 1.3 is thread-based on the Windows platform, it is a process-based Web server on all UNIX and Linux platforms. That means, it implements the *multi-process, single-thread* process model: for each incoming request, a new child process is created or requested from a pool to handle it.

However, Apache 2.0 is now a fully thread-based Web server on all platforms. This gives you the following advantages:

- Each request does not require its *own* httpd process anymore, and less memory is needed.
- Overall performance improves because in most cases new httpd processes do not need to be created.
- The plug-in load-balancing and failover algorithms work more efficiently in a multi-threaded HTTP server. If one plug-in thread marks an application server unavailable, all other connection threads of the plug-in within the same process will share that knowledge, and will not try to connect to this particular application server again before the `RetryInterval` has elapsed. See 6.10, “WebSphere plug-in behavior” on page 309 for information about the `RetryInterval` parameter and plug-in load balancing and failover issues.

Note: On the UNIX platform, Apache 2.0 also allows you to configure more than one process to be started. This means that the thread model is then changed to multi-process, multi-thread.

► The *mod_deflate* module

This module allows supporting browsers to request that content be compressed before delivery. It provides the *Deflate* output filter that lets output from the server be compressed before it is sent to the client over the network. Some of the most important benefits of using the *mod_deflate* module are:

- Saves network bandwidth during data transmission
- Shortens data transmission time
- Generally improves overall performance

Detailed information about configuring and using *mod_deflate* can be found at

http://httpd.apache.org/docs-2.0/mod/mod_deflate.html

► Request and response filtering

Apache modules may now be written as filters which act on the stream of content as it is delivered to or from the server.

Single-processing versus multi-processing

Apache 2.0 achieves efficient support of different operating systems by implementing a Multi-Processing Modules (MPM) architecture, allowing it, for example, to use native networking features instead of going through an emulation layer in version 1.3. For detailed information about MPM refer to the Apache HTTP Server documentation found at:

<http://httpd.apache.org/docs-2.0/mpm.html>

MPMs are chosen at compile time and differ for each operating system, which implies that the Windows version uses a different MPM module than the AIX or Linux version. The default MPM for Windows is `mpm_winnt`, whereas the default module for AIX is `mpm_worker`. For a complete list of available MPMs, refer to the Apache MPM documentation URL above. To identify which MPM compiled into an Apache 2.0 Web server, run the **`apachectl -l`** command, which prints out the module names. Look for a module name `worker`, or a name starting with the `mpm` prefix (see Example 17-4).

Example 17-4 Listing of compiled in modules for IBM HTTP Server V6 on AIX

```
# ./apachectl -l
Compiled in modules:
  core.c
  worker.c
  http_core.c
  mod_suexec.c
  mod_so.c
```

► **`mpm_winnt` module**

This Multi-Processing Module is the default for the Windows operating systems. It uses a single control process which launches a single child process which in turn creates all the threads to handle requests.

► **`mpm_worker` module**

This Multi-Processing Module implements a hybrid multi-process multi-threaded server. This is the default module for AIX. By using threads to serve requests, it is able to serve a large number of requests with less system resources than a process-based server. Yet it retains much of the stability of a process-based server by keeping multiple processes available, each with many threads.

Tuning IBM HTTP Server V6

This section gives you configuration tips for the UNIX and Windows platform that provide a good starting point for Web server tuning, and an explanation of the

new configuration directives used. Keep in mind that every system and every site has different requirements, so make sure to adapt these settings to your needs! See Example 17-5 for a UNIX sample configuration and Example 17-6 on page 1019 for a Windows sample configuration.

► **ThreadsPerChild**

Each child process creates a fixed number of threads as specified in the `ThreadsPerChild` directive. The child creates these threads at startup and never creates more. If using an MPM like `mpm_winnt`, where there is only one child process, this number should be high enough to handle the entire load of the server. If using an MPM like `worker`, where there are multiple child processes, the total number of threads should be high enough to handle the common load on the server.

► **ThreadLimit**

This directive sets the maximum configured value for `ThreadsPerChild` for the lifetime of the Apache process. `ThreadsPerChild` can be modified during a restart up to the value of this directive.

► **MaxRequestsPerChild**

This directive controls after how many requests a child server process is recycled and a new one is launched.

► **MaxClients**

This controls the maximum total number of threads that may be launched.

► **StartServers**

The number of processes that will initially be launched is set by the `StartServers` directive.

► **MinSpareThreads and MaxSpareThreads**

During operation, the total number of idle threads in all processes will be monitored, and kept within the boundaries specified by `MinSpareThreads` and `MaxSpareThreads`.

► **ServerLimit**

The maximum number of processes that can be launched is set by the `ServerLimit` directive.

Example 17-5 A sample configuration for the UNIX platform

```
<IfModule worker.c>
ServerLimit 1
ThreadLimit 2048
StartServers 1
MaxClients 1024
MinSpareThreads 1
```

```
MaxSpareThreads 1024
ThreadsPerChild 1024
MaxRequestsPerChild 0
</IfModule>
```

Attention: Using a `ThreadsPerChild` value greater than 512 is not recommended on the Linux and Solaris platform. If 1024 threads are needed, the recommended solution is to increase the `ServerLimit` value to 2 to launch two server process with 512 threads each.

Example 17-6 A sample configuration for the Windows platform

```
<IfModule mpm_winnt.c>
ThreadsPerChild 2048
MaxRequestsPerChild 0
</IfModule>
```

Access logs

All incoming HTTP requests are logged here. Logging degrades performance because of the (possibly significant) I/O overhead.

To turn logging on or off, edit the IBM HTTP Server `httpd.conf` file, located in the directory `<IBM HTTP Server Home>/conf`. Search for a line with the text `CustomLog`. Comment out this line, then save and close the `httpd.conf` file. Stop and restart the IBM HTTP Server. By default, logging is enabled, but for better performance it is recommended that you disable the access logs.

Sun Java System Web server, Enterprise Edition

(formerly Sun ONE Web Server, formerly iPlanet)

The default configuration of the Sun Java System Web server, Enterprise Edition provides a single-process, multi-threaded server. Refer to the InfoCenter article “Configuring the Sun Java System Web Server” for tips.

The InfoCenter is available at:

<http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp>

Microsoft IIS

The Web server has several properties that dramatically affect the performance of the application server, such as IIS permission properties. Refer to the InfoCenter article “Configuring Microsoft Internet Information Services (IIS)” for additional information.

17.5.9 Dynamic Cache Service

The Dynamic Cache Service improves performance by caching the output of servlets, commands and Java Server Pages (JSP) files. WebSphere Application Server consolidates several caching activities, including servlets, Web services, and WebSphere commands into one service called the dynamic cache. These caching activities work together to improve application performance, and share many configuration parameters, which are set in an application server's dynamic cache service.

The dynamic cache works within an application server Java Virtual Machine (JVM), intercepting calls to cacheable objects, for example through a servlet's `service()` method or a command's `execute()` method, and either stores the object's output to or serves the object's content from the dynamic cache. Because J2EE applications have high read/write ratios and can tolerate small degrees of latency in the currency of their data, the dynamic cache can create an opportunity for significant gains in server response time, throughput, and scalability.

See 10.4, “Using WebSphere dynamic cache service” on page 515 for an in-depth discussion of dynamic caching, and 10.4.1, “Installing Dynamic Cache Monitor” on page 516 on using the dynamic cache monitor.

17.5.10 Security settings

This section discusses how various settings related to security affect performance. Refer to *WebSphere Application Server V6: Security Handbook*, SG24-6316 for more information about WebSphere Security.

When evaluating security for your environment, always keep the following steps in mind:

1. Analyze your security needs regarding authentication, authorization, and communication paths over which this information is exchanged.
2. Turn off security where you do not need it.
3. Make sure you do not sacrifice security for the sake of performance.

Disabling security

Security is a global setting. When security is enabled, performance may be decreased by up to 20%.

In the Administrative Console, select **Security -> Global Security**. The **Enable global security** and **Enforce Java 2 security** check boxes control global security settings.

Fine-tune the security cache timeout for the environment

If WebSphere Application Server security is enabled, the security cache timeout can influence performance. The timeout parameter specifies how often to refresh the security-related caches.

Security information pertaining to beans, permissions, and credentials is cached. When the cache timeout expires, all cached information becomes invalid. Subsequent requests for the information result in a database lookup. Sometimes, acquiring the information requires invoking a Lightweight Directory Access Protocol (LDAP)-bind or native authentication. Both invocations are relatively costly operations for performance.

Determine the best trade-off for the application by looking at usage patterns and security needs for the site.

Use the Administrative Console to change this value. To do so, select **Security** -> **Global Security**. Enter an appropriate value in seconds in the Cache Timeout field. The default is 600 seconds.

17.5.11 Tuning Secure Sockets Layer

The following are two types of Secure Sockets Layer (SSL) performance:

- ▶ Handshake
- ▶ Bulk encryption/decryption

Overview of handshake and bulk encryption and decryption

When an SSL connection is established, an SSL handshake occurs. After a connection is made, SSL performs bulk encryption and decryption for each read/write. The performance cost of an SSL handshake is much larger than that of bulk encryption and decryption.

How to enhance SSL performance

In order to enhance SSL performance, the number of individual SSL connections and handshakes must be decreased.

Decreasing the number of connections increases performance for secure communication through SSL connections, as well as non-secure communication through simple TCP connections. One way to decrease individual SSL connections is to use a browser that supports HTTP 1.1. Decreasing individual SSL connections could be impossible for some users if they cannot upgrade to HTTP 1.1.

Another common approach is to decrease the number of connections (both TCP and SSL) between two WebSphere Application Server components. The

following guidelines help to ensure the HTTP transport channel of the application server is configured so that the Web server plug-in does not repeatedly reopen new connections to the application server:

- ▶ The maximum number of requests per keep-alive connection can also be increased. The default value is 100, which means the application server will close the connection from the plug-in after 100 requests. The plug-in would then have to open a new connection. The purpose of this parameter is to prevent denial of service attacks when connecting to the application server and continuously sending requests in order to tie up threads in the application server.
- ▶ Use a hardware accelerator if the system performs several SSL handshakes. Hardware accelerators currently supported by WebSphere Application Server only increase the SSL handshake performance, not the bulk encryption/decryption. An accelerator typically only benefits the Web server because Web server connections are short-lived. All other SSL connections in WebSphere Application Server are long-lived.
- ▶ Use an alternative cipher suite with better performance.

The performance of a cipher suite is different with software and hardware. Just because a cipher suite performs better in software does not mean a cipher suite will perform better with hardware. Some algorithms are typically inefficient in hardware (for example, DES and 3DES). However, specialized hardware can provide efficient implementations of these same algorithms.

The performance of bulk encryption and decryption is affected by the cipher suite used for an individual SSL connection.

17.5.12 Object Request Broker (ORB)

Several settings are available for controlling internal Object Request Broker (ORB) processing. You can use these to improve application performance in the case of applications containing enterprise beans.

You can change these settings for the default server or any application server configured in the administrative domain from the Administrative Console.

Pass by reference

For EJB 1.1 beans, the EJB 1.1 specification states that method calls are to be Pass by value. For every remote method call, the parameters are copied onto the stack before the call is made. This can be expensive. The Pass by reference, which passes the original object reference without making a copy of the object, can be specified.

For EJB 2.0 beans, interfaces can be local or remote. For local interfaces, method calls are Pass by reference, by default.

If the EJB client and EJB server are installed in the same WebSphere Application Server instance, and the client and server use remote interfaces, specifying Pass by reference can improve performance up to 50%.

Please note that Pass by reference helps performance only when non-primitive object types are being passed as parameters. Therefore, int and floats are always copied, regardless of the call model.

Important: Pass by reference can be dangerous and can lead to unexpected results. If an object reference is modified by the remote method, the change might be seen by the caller.

The use of this option for enterprise beans with remote interfaces violates EJB Specification, Version 2.0 section 5.4. Object references passed to EJB methods or to EJB home methods are not copied and can be subject to corruption.

In Example 17-7, a reference to the same `MyPrimaryKey` object passes into WebSphere Application Server with a different ID value each time. Running this code with Pass by reference enabled causes a problem within the application server because multiple enterprise beans are referencing the same `MyPrimaryKey` object. To avoid this problem, set the `com.ibm.websphere.ejbcontainer.allowPrimaryKeyMutation` system property to true when Pass by reference is enabled. Setting `allowPrimaryKeyMutation` to true causes the EJB container to make a local copy of the `PrimaryKey` object. As a result, however, a small portion of the performance advantage of setting Pass by reference is lost.

Example 17-7 Pass by reference problem demonstration

```
Iterator iterator = collection.iterator();
MyPrimaryKey pk = new MyPrimaryKey();
while (iterator.hasNext()) {
    pk.id = (String) iterator.next();
    MyEJB myEJB = myEJBHome.findByPrimaryKey(pk);
}
```

Use the Administrative Console to set `allowPrimaryKeyMutation`:

1. Select **Servers -> Application servers -> <AppServer_Name>**.
2. Then, select **Container Services -> ORB Service**.
3. Select **Custom Properties**.
4. Create a new property by clicking **New**.

5. Assign the new property the name **`com.ibm.websphere.ejbcontainer.allowPrimaryKeyMutation`** and a value of **true**.
6. Click **OK** and save the changes.
7. Stop and restart the application server.

As a general rule, any application code that passes an object reference as a parameter to an enterprise bean method or to an EJB home method must be scrutinized to determine if passing that object reference results in loss of data integrity or in other problems.

Use the Administrative Console to set this value:

1. Select **Servers -> Application servers -> <AppServer_Name>**.
2. Then, select **Container Services -> ORB Service**.
3. Select the check box **Pass by reference**.
4. Click **OK** and **Apply** to save the changes.
5. Stop and restart the application server.

If you use command line scripting, the full name of this system property is `com.ibm.CORBA.iiop.noLocalCopies`.

The default is Pass by value for remote interfaces and Pass by reference for EJB 2.0 local interfaces.

If the application server expects a large workload for enterprise bean requests, the ORB configuration is critical. Take note of the following properties.

com.ibm.CORBA.ServerSocketQueueDepth

This property corresponds to the length of the TCP/IP stack listen queue and prevents WebSphere Application Server from rejecting requests when there is no space in the listen queue.

If there are many simultaneous clients connecting to the server-side ORB, this parameter can be increased to support the heavy load up to 1000 clients. The default value is 50.

To set the property (in our example we set it to 200), follow these steps:

1. Select **Servers -> Application servers -> <AppServer_Name>**.
2. Then, select **Container Services -> ORB Service**.
3. Select **Custom Properties**.

4. Click **New** and create a new name/value pair with the name **com.ibm.CORBA.ServerSocketQueueDepth** and the value **200**.
5. Click **OK** and **Apply** to save the changes.
6. Stop and restart the application server.

Connection cache maximum

This property has two names and corresponds to the size of the ORB connection table. The property sets the standard for the number of simultaneous ORB connections that can be processed.

If there are many simultaneous clients connecting to the server-side ORB, this parameter can be increased to support the heavy load up to 1000 clients. The default value is 240.

If you use command line scripting, the full name of this system property is `com.ibm.CORBA.MaxOpenConnections`.

Use the Administrative Console to set this value:

1. Select **Servers -> Application servers -> <AppServer_Name>**.
2. Then, select **Container Services -> ORB Service**.
3. Update the Connection cache maximum field and click **OK**.
4. Click **Apply** to save the changes then restart the application server.

ORB thread pool size

Refer to “EJB container” on page 986 for more information.

17.5.13 XML parser selection

Add XML parser definitions to the `jaxp.properties` file and `xerces.properties` file found in the `<WAS_HOME>/jre/lib` directory to help facilitate server startup. The `XMLParserConfiguration` value might have to be changed as new versions of Xerces are provided.

In both files, insert the lines shown in Example 17-8.

Example 17-8 XML parser definitions

```
javax.xml.parsers.SAXParserFactory=org.apache.xerces.jaxp.SAXParserFactoryImpl
javax.xml.parsers.DocumentBuilderFactory=org.apache.xerces.jaxp.DocumentBuilderFactoryImpl
org.apache.xerces.xni.parser.XMLParserConfiguration=org.apache.xerces.parsers.StandardParserConfiguration
```

17.5.14 DB2 tuning

DB2 has many parameters that can be configured to optimize database performance. For complete DB2 tuning information, refer to the *DB2 UDB Administration Guide: Performance*.

DB2 logging

DB2 has corresponding log files for each database. Performance improvements can be gained by setting the log files on a different hard drive from the database files. Refer to “AIX with DB2” on page 1013 for more information.

DB2 configuration advisor

Located in the DB2 Control Center, this advisor calculates and displays recommended values for the DB2 buffer pool size, the database and database manager configuration parameters, with the option of applying these values. See more information about the advisor in the online help facility within the Control Center.

Use TCP sockets for DB2 on Linux

On Linux platforms, whether the DB2 server resides on a local machine with WebSphere Application Server or on a remote machine, configure the DB2 application databases to use TCP sockets for communications with the database.

The directions for configuring DB2 on Linux can be found in the WebSphere Application Server installation documentation for the various operating systems. This document specifies setting DB2COMM for TCP/IP and corresponding changes required in the /etc/services file.

The default is to use shared memory for local databases but it is recommended that you change the specification for the DB2 application databases and for any session databases from shared memory to TCP sockets.

DB2 MaxAppls and DB2 MaxAgents

When configuring the data source settings for the databases, confirm the DB2 MaxAppls setting is greater than the maximum number of connections for the data source. If you are planning to use multiple cluster members, set the MaxAppls value as the maximum number of connections multiplied by the number of cluster members.

The same relationship applies to the session manager number of connections. The MaxAppls setting must be equal to or greater than the number of connections. If you are using the same database for session and data sources,

set the MaxAppls value as the sum of the number of connection settings for the session manager and the data sources.

Refer to “Connection pool size” on page 983 for more information about minimum and maximum number of connections.

DB2 buffpage

Buffpage is a database configuration parameter. It defines the amount of memory that will be allocated for a new defined bufferpool, if you omit it in the DDL. A bufferpool is a memory storage area where database pages containing table rows or index entries are temporarily read and changed. The purpose of the bufferpool is to improve database system performance. Data can be accessed much faster from memory than from disk.

Refer to the WebSphere InfoCenter section “DB2 tuning parameters” for more information about how to configure this parameter.

DB2 query optimization level

When a database query is executed in DB2, various methods are used to calculate the most efficient access plan. The query optimization level parameter sets the amount of work and resources that DB2 puts into optimizing the access plan. The range is from zero to 9.

An optimization level of 9 causes DB2 to devote a lot of time and all of its available statistics to optimizing the access plan.

The optimization level is set on individual databases and can be set with either the command line or with the DB2 Control Center. Static SQL statements use the optimization level specified on the **prep** and **bind** commands. If the optimization level is not specified, DB2 uses the default optimization as specified by the `dft_queryopt` parameter. Dynamic SQL statements use the optimization class specified by the current query optimization special register, which is set using the SQL Set statement. For example, the following statement sets the optimization class to 1:

```
Set current query optimization = 1
```

If the current query optimization register has not been set, dynamic statements will be bound using the default query optimization class.

The default value is 5. It is recommended that you set the optimization level for the needs of the application. High levels should only be used when there are very complicated queries.

DB2 reorgchk

The performance of the SQL statements can be impaired after many updates, deletes, or inserts have been made. Performance can be improved by obtaining the current statistics for the data and rebinding.

Use the following DB2 command to issue runstats on all user and system tables for the database you are currently connected to:

```
db2 reorgchk update statistics on table all
```

You should then rebind packages using the **bind** command.

In order to see if runstats has been done, issue the following command on DB2 CLP:

```
db2 -v "select tbname, nleaf, nlevels, stats_time from sysibm.sysindexes"
```

If no runstats has been done, nleaf and nlevels will be filled with -1 and stats_time will have an empty entry "-". If runstats was done already, the real-time stamp when the runstats was completed will also be displayed under stats_time. If you think the time shown for the previous runstats is too old, execute runstats again.

DB2 MinCommit

This parameter allows delayed writing of log records to a disk until a minimum number of commits have been performed, reducing the database manager overhead associated with writing log records. For example, if MinCommit is set to 2, a second commit would cause output to the transaction log for the first and second commits. The exception occurs when a one-second timeout forces the first commit to be output if a second commit does not come along within one second. In test applications, up to 90% of the disk input and output was related to the DB2 transaction log. Changing MinCommit from 1 to 2 reduced the results to 45%.

Try to adjust this parameter if the disk input/output wait is more than 5% and there is DB2 transaction log activity from multiple sources. When a lot of activity occurs from multiple sources, it is less likely that a single commit will have to wait for another commit (or the one-second timeout).

Do not adjust this parameter if you have an application with a single thread performing a series of commits (each commit could hit the one-second delay).

To view the current value for a particular database follow these steps:

- ▶ Issue the DB2 command **get db cfg for <dbname>** (where <dbname> is the name of the application database) to list database configuration parameters.
- ▶ Look for "Group commit count (MINCOMMIT)".

- ▶ Set a new value by issuing the DB2 command **update db cfg for <dbname> using mincommit n** (where n is a value between 1 and 25 inclusive).

The new setting takes effect immediately.

The following are several metrics that are related to DB2 MinCommit:

- ▶ The disk input/output wait can be observed on AIX with the command **vmstat 5**. This shows statistics every 5 seconds. Look for the *wa* column under the CPU area.
- ▶ The percentage of time a disk is active can be observed on AIX with the command **iostat 5**. This shows statistics every 5 seconds. Look for the *%tm_act* column.
- ▶ The DB2 command **get snapshot for db on <dbname>** (where <dbname> is the name of the application database) shows counters for log pages read and log pages written.

The default value is 1. It is recommended that you set MinCommit to 1 or 2 (if the circumstance permits).

17.5.15 Additional reference materials

- ▶ IBM WebSphere Application Server Library, including monitoring and troubleshooting documentation, are found at:
<http://www.ibm.com/software/webservers/appserv/library/index.html>
- ▶ WebSphere Application Server White papers found at:
<http://www.ibm.com/support/search.wss?rs=180&tc=SSEQTP&dc=DB100>
- ▶ iSeries performance documents, including *WebSphere Application Server for iSeries Performance Considerations* and links to the PTDV tool, Workload Estimator tool, and other documents are found at:
<http://www.ibm.com/servers/eserver/iseries/software/websphere/wsappserver/product/PerformanceConsiderations.html>
- ▶ “J2EE Application Development: One or many applications per application server?” (Nov. 2002) found at:
http://www.ibm.com/developerworks/websphere/techjournal/0211_alcott/alcott.html
- ▶ “Handling Static content in WebSphere Application Server” (Nov. 2002) found at:
http://www.software.ibm.com/wsdd/techjournal/0211_brown/brown.html

- The WebSphere Application Server zone at WebSphere Developer Domain at:

<http://www.ibm.com/developerworks/websphere/zones/was/>



Part 7

Appendixes

Sample URL rewrite servlet

This appendix describes how to set up an example to test session management using URL rewrites.

Setting up the servlet

The class `SessionSampleURLRewrite` has been provided since none of the standard samples uses URL rewriting to manage sessions. This is because using URL rewrites requires additions to the normal servlet code.

Source code

Example A-1 lists the Java source for the class. You can either compile and package this class yourself or you can use the pre-compiled class file packaged in `urptest.war`. You can find information about how to download `urptest.war` in Appendix B, “Additional material” on page 1037.

Example: A-1 SessionSampleURLRewrite class source code

```
public class SessionSampleURLRewrite extends HttpServlet {
    public void doGet (HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        // Step 1: Get the Session object

        boolean create = true;
        HttpSession session = request.getSession(create);

        // Step 2: Get the session data value

        Integer ival = (Integer)
            session.getAttribute ("sessiontest.counter");
        if (ival == null) ival = new Integer (1);
        else ival = new Integer (ival.intValue () + 1);
        session.setAttribute ("sessiontest.counter", ival);

        // Step 3: Rewrite the session ID onto the URL

        String contextPath = request.getContextPath();
        String servletName = request.getServletPath();
        String encodeString = contextPath + servletName;
        String encodedURL = response.encodeURL(encodeString);

        // Step 4: Output the page

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>Session Tracking Test</title></head>");
    }
}
```

```

        out.println("<body>");
        out.println("<h1>Session Tracking Test</h1>");
        out.println ("You have hit this page " + ival + " times" + "<br>");

        if (!session.isNew()) {
            out.println("<p>");
            if(request.isRequestedSessionIdFromURL())
                out.print("Your requested session ID was found in a rewritten
URL");
            else
                out.print("Your requested session ID was NOT found in a rewritten
URL");
        }

        // Use the rewritten URL as the link, You must enable URL Rewriting in the
        session Manager and
        // disable cookies in either the Session Manager or the browser

        out.print("<p>");
        out.print("<a href=\"");
        out.print(encodedURL);
        out.println("\>Request this servlet again using the rewritten URL</a>");

        out.println("</body></html>");
    }
}

```

Steps to install SessionSampleURLRewrite servlet

Please refer to 8.7.2, “Install BeenThere” on page 427 for detailed instructions on how to install an enterprise application.

Installing the urltest Web module

In summary, these are the steps to install urltest.war:

1. Locate urltest.war as described in Appendix B, “Additional material” on page 1037 and store it on your workstation or on your server.
2. Open the WebSphere Administrative Console and select **Applications -> Install New Application**.
3. In the Preparing for the application installation window:
 - a. Browse to the urltest.war Web module archive (Local path or Server path).

- b. Enter a context root, such as `/urltest`.
- c. Click **Next**.
4. Leave all defaults on the second Preparing for the application installation window and click **Next** again. If you get security warnings, just click **Continue**.
5. If desired, change the name of the application (default = `urltest_war`) in Step 1 of the Install New Application procedure. Click **Next**.
6. On the Step 2: Map modules to servers window, from the Clusters and Servers selection box, choose the application server and the HTTP server on you wish to install `urltest.war`. Then check the box for the Module `urltest` and click **Apply**. Clicking **Next** brings you to Step 3.
7. Map the Web Module `urltest` to the desired Virtual host. Click **Next** once again.
8. Verify the installation on the Step 4: Summary window and click **Finish** to install the `urltest` Web module.
9. Save the changes to the Master Configuration.
10. To start the `urltest_war` enterprise application, locate `urltest_war` in the Enterprise Applications view. Select the application and click **Start**.
11. If you did not configure your Web Servers to automatically regenerate and propagate the plug-in, you need to do it now (if needed, refer to 8.7.3, "Regenerate Web server plug-in" on page 431 for instructions on how to do so).

You can now access the URL rewrite sample application using:

`http://<your_http_server>:<port>/<your_context_root>/urltest`

For example:

`http://http1/urltest/urltest`

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/SG246392>

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG24-6392.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

<i>File name</i>	<i>Description</i>
urltest.war	URL Rewrite Example Web module
BeenThere.ear	Original BeenThere application code
Trade6Redirector.zip	Trade 6 back-end application for WebSphere MQ scenario
BeenThereSIB.zip	BeenThere modified for default messaging provider

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space:	10 MB
Operating System:	Windows, AIX, Solaris, Linux, OS/400®
Processor:	500 MHz Pentium®, RS/6000®, iSeries, Sparc
Memory:	384 MB RAM minimum

How to use the Web material

- ▶ Create a subdirectory (folder) on your workstation or your server, and download the urltest.war file into this folder.
- ▶ Create a subdirectory (folder) on a Windows system and download the BeenThere.ear file into this folder. This Enterprise Application Resource is all you need to install the BeenThere application on your applications server(s).
- ▶ Create a subdirectory (folder) on a Windows system and download the Trade6Redirector.zip file into this folder. Unzip the files. The zip file consists of two files:

- Trade6Redirector.ear
- Instructions for setup of TradeRedirector.txt

Follow the install instructions from the text file.

- ▶ Create a subdirectory (folder) on a Windows system and download the BeenThereSIB.zip file into this folder. Unzip the files. The zip-file consists of two files:
 - BeenThere.ear (this is the BeenThere version that was modified for the default messaging provider!)
 - BeenThereDocumentation.zip

Then unzip the BeenThereDocumentation.zip archive.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 1047. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451
- ▶ *IBM WebSphere V6 Planning and Design Handbook*, SG24-6446
- ▶ *WebSphere Application Server V6: Security Handbook*, SG24-6316
- ▶ *Rational Application Developer V6 Programming Guide*, SG24-6449
- ▶ *WebSphere Version 6 Web Services Handbook Development and Deployment*, SG24-6461
- ▶ *WebSphere Application Server Network Deployment V6: High availability solutions*, SG24-6688
- ▶ *IBM WebSphere V5.1 Performance, Scalability, and High Availability, WebSphere Handbook Series*, SG24-6198-01
- ▶ *Patterns: Self-Service Application Solutions Using WebSphere V5.0*, SG24-6591
- ▶ *WebSphere Studio 5.1.2 JavaServer Faces and Service Data Objects*, SG24-6361
- ▶ *Planning for the Installation and Rollout of WebSphere Studio Application Monitor 3.1*, SG24-7072
- ▶ *Installing WebSphere Studio Application Monitor V3.1*, SG24-6491
- ▶ *End-to-End e-business Transaction Management Made Easy*, SG24-6080

Other publications

These publications are also relevant as further information sources:

- ▶ *Load Balancer Administration Guide Version 6.0*, GC31-6858
- ▶ *Caching Proxy Administration Guide Version 6.0*, GC31-6857
- ▶ *Concepts, Planning, and Installation for Edge Components Version 6.0*, GC31-6855
- ▶ IBM JVM Diagnostics Guides
<http://www.ibm.com/developerworks/java/jdk/diagnosis/>
- ▶ “Performance Testing Protocol for WebSphere Application Server-based Applications”
http://www7b.software.ibm.com/wsdd/techjournal/0211_polozoff/polozoff.html
- ▶ Stacy Joines, et.al., *Performance Analysis for Java Web Sites*, Addison-Wesley, September 2002, ISBN 0201844540
- ▶ Kareem Yusuf, Ph.D, *Enterprise Messaging Using JMS and IBM WebSphere*, IBM Press, 2004, ISBN 0131468634
- ▶ Floyd Marinescu, *EJB Design Patterns: Advanced Patterns, Processes, and Idioms*, Wiley, 2002, ISBN 0471208310

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ WebSphere Application Server InfoCenter
<http://www.ibm.com/software/webservers/appserv/infocenter.html>
- ▶ Edge Components InfoCenter
<http://www.ibm.com/software/webservers/appserv/ecinfocenter.html>
- ▶ IBM WebSphere Application Server
<http://www.ibm.com/software/webservers/appserv>
- ▶ IBM WebSphere Application Server hardware and software requirements
<http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html>
- ▶ IBM WebSphere Application Server support page
<http://www.ibm.com/software/webservers/appserv/was/support/>
- ▶ IBM WebSphere Edge Server
<http://www.ibm.com/software/webservers/edgeserver>

- ▶ IBM DB2 UDB hardware and software requirements
<http://www.ibm.com/software/data/db2/udb/sysreqs.html>
- ▶ HTTP server configuration at IBM HTTP Server InfoCenter
<http://www.ibm.com/software/webservers/httpservers/library.html>
- ▶ IBM WebSphere Application Server Library
<http://www.ibm.com/software/webservers/appserv/library/index.html>
- ▶ WebSphere Application Server Development white papers
<http://www.ibm.com/software/webservers/appserv/whitepapers.html>
- ▶ WebSphere Application Server zone at WebSphere Developer Domain
<http://www7b.boulder.ibm.com/wsdd/zones/was/>
- ▶ WebSphere performance Web site, including Trade 6 code download
<http://www.ibm.com/software/webservers/appserv/was/performance.html>
- ▶ IBM @server iSeries performance documents
<http://www.ibm.com/servers/eserver/iseries/software/websphere/wsappserver/product/PerformanceConsiderations.html>
- ▶ GUI tool for building cache policy files
<http://www.alphaworks.ibm.com/tech/cachepolicyeditor>
- ▶ WebSphere Developer Domain
<http://www.software.ibm.com/wsdd/>
- ▶ IBM developerWorks
<http://www.ibm.com/developerworks/>
- ▶ IBM alphaWorks
<http://www.alphaworks.ibm.com>
- ▶ Page Detailer information and download
<http://www.alphaworks.ibm.com/tech/pagedetailer>
- ▶ IBM Tivoli Monitoring for Web Infrastructure
<http://www.ibm.com/software/tivoli/products/monitor-web/>
- ▶ IBM Tivoli Monitoring for Transaction Performance
<http://www.ibm.com/software/tivoli/products/monitor-transaction/>
- ▶ List of IBM's Business Partners that offer performance monitoring tools compliant with WebSphere Application Server
http://www.ibm.com/software/webservers/pw/dhtml/wsperformance/performance_bpsolutions.html

- ▶ Java Performance Tuning Web site
<http://www.javaperformancetuning.com/>
- ▶ Design for Scalability - An Update
<http://www.ibm.com/developerworks/websphere/library/techarticles/hvws/scalability.html>
- ▶ Technote TIPS0223 (WebSphere Application Server V5: Separating Static and Dynamic Web Content):
<http://publib-b.boulder.ibm.com/Redbooks.nsf/RedbookAbstracts/tips0223.html?Open>
- ▶ Caching Technologies for Web Applications
<http://www.almaden.ibm.com/u/mohan/>
- ▶ Exploiting Dynamic Caching in WebSphere Application Server 5.0, Part 1 & 2
<http://www.e-promag.com/eparchive//index.cfm?fuseaction=viewarticle&ContentID=3623&publicationid=19&PageView=Search&channel=2>
<http://www.e-promag.com/eparchive//index.cfm?fuseaction=viewarticle&ContentID=4409&publicationid=19&PageView=Search&channel=2>
- ▶ Monitoring Performance with WebSphere
<http://www.e-promag.com/eparchive//index.cfm?fuseaction=viewarticle&ContentID=1492&publicationid=13&PageView=Search&channel=2>
- ▶ Hints on Running a High-Performance Web Server - Tuning IBM IHS
<http://www.ibm.com/software/webservers/htpservers/doc/v136/misc/perf.html>
- ▶ Enhancements and Changes in J2SE 1.4.1 Platform
http://java.sun.com/products/archive/j2se/1.4.1_07/changes.html
- ▶ Reducing Garbage Collection Times and Sizing Memory
<http://developers.sun.com/techtopics/mobility/midp/articles/garbage/>
- ▶ Improving Java Application Performance and Scalability by Reducing Garbage Collection Times and Sizing Memory Using JDK 1.4.1
<http://developers.sun.com/techtopics/mobility/midp/articles/garbagecollection2/>
- ▶ Fine-tuning Java garbage collection performance
<http://www.ibm.com/developerworks/ibm/library/i-gctroub/>
- ▶ Garbage collection in the 1.4.1 JVM
<http://www.ibm.com/developerworks/java/library/j-jtp11253/>
- ▶ A brief history of garbage collection
<http://www.ibm.com/developerworks/java/library/j-jtp10283/>

- ▶ Sensible Sanitation: Understanding the IBM Java Garbage Collection, Parts 1 and 2
<http://www.ibm.com/developerworks/ibm/library/i-garbage1/>
<http://www.ibm.com/developerworks/ibm/library/i-garbage2/>
- ▶ Sensible Sanitation: Understanding the IBM Java Garbage Collection Part 3: verbosegc and command-line parameters
<http://www.ibm.com/developerworks/library/i-garbage3.html>
- ▶ Mash that trash - Incremental compaction in the IBM JDK Garbage Collector
<http://www.ibm.com/developerworks/ibm/library/i-incrcomp/>
- ▶ J2EE Application Development: One or many applications per application server?
http://www.software.ibm.com/wsdd/techjournal/0211_alcott/alcott.html
- ▶ Handling Static content in WebSphere Application Server
http://www.software.ibm.com/wsdd/techjournal/0211_brown/brown.html
- ▶ JavaServer Pages 2.0 Specification
<http://jcp.org/aboutJava/communityprocess/final/jsr152/index.html>
- ▶ Struts, an open-source MVC implementation
<http://www.ibm.com/developerworks/ibm/library/j-struts/>
- ▶ Struts home page
<http://struts.apache.org/>
- ▶ Integrating Struts, Tiles, and JavaServer Faces
<http://www.ibm.com/developerworks/java/library/j-integrate/>
- ▶ New to SOA and Web services
<http://www.ibm.com/developerworks/webservices/newto/index.html>
- ▶ Developing and Deploying Command Caching with WebSphere Studio V5
http://www.ibm.com/developerworks/websphere/registered/tutorials/0306_mcguinnes/mcguinnes.html
- ▶ JSR 235: Service Data Objects
<http://www.jcp.org/en/jsr/detail?id=235>
- ▶ Introduction to Service Data Objects
<http://www.ibm.com/developerworks/java/library/j-sdo/>
- ▶ Log4J
<http://jakarta.apache.org/log4j/>

- ▶ IBM WebSphere Developer Technical Journal: Writing PMI applications using the JMX interface
http://www.ibm.com/developerworks/websphere/techjournal/0402_qiao/0402_qiao.html
- ▶ IBM WebSphere Developer Technical Journal: Writing a Performance Monitoring Tool Using WebSphere Application Server's Performance Monitoring Infrastructure API
http://www.ibm.com/developerworks/websphere/techjournal/0202_rangaswamy/rangaswamy.html
- ▶ WebSphere Performance Diagnostics - Going beyond the Metrics
<http://www.sys-con.com/websphere/article.cfm?id=207>
- ▶ Design for Performance: Analysis of Download Times for Page Elements Suggests Ways to Optimize
<http://www.ibm.com/developerworks/websphere/library/techarticles/hvws/performance.html>
- ▶ Implementing a Highly Available Infrastructure for WebSphere Application Server Network Deployment, Version 5.0 without Clustering
http://www.ibm.com/developerworks/apps/transform.wss?URL=/developerworks/websphere/library/techarticles/0304_alcott/alcott.xml&xslURL=/developerworks/websphere/xsl/document.xsl&format=one-column
- ▶ Server Clusters For High Availability in WebSphere Application Server Network Deployment Edition 5.0
<http://www.ibm.com/support/docview.wss?uid=swg27002473>
- ▶ JMS Application Architectures
<http://www.theserverside.com/articles/article.tss?l=JMSArchitecture>
- ▶ JMS 1.1 simplifies messaging with unified domains”, found at:
<http://www.ibm.com/developerworks/java/library/j-jms11/>
- ▶ JMS Topologies and Configurations with WebSphere Application Server and WebSphere Studio Version 5
http://www.ibm.com/developerworks/websphere/library/techarticles/0310_barciabarcia.html
- ▶ Learning About JMS Messages
<http://www.phptr.com/articles/article.asp?p=170722>
- ▶ Supportpacs about JMS performance with WebSphere MQ
<http://www.ibm.com/software/integration/support/supportpacs/product.html>
http://www.ibm.com/support/docview.wss?rs=203&uid=swg24006854&loc=en_US&cs=utf-8&lang=en

http://www.ibm.com/support/docview.wss?rs=203&uid=swg24006902&loc=en_US&cs=utf-8&lang=en

- ▶ J2EE Connector Architecture
<http://java.sun.com/j2ee/connector/download.html>
- ▶ Java Security, JAAS, JCE and JSSE
<http://java.sun.com/security>
- ▶ Apache Software Foundation
<http://www.apache.org>
- ▶ Apache HTTP Server source-code
<http://httpd.apache.org/download.cgi>
- ▶ Apache 2.0 Feature list
http://httpd.apache.org/docs-2.0/new_features_2_0.html
- ▶ Apache HTTP Server documentation
<http://httpd.apache.org/docs-2.0/mpm.html>
- ▶ Akamai
<http://www.akamai.com>
- ▶ Speedera
<http://www.speedera.com>
- ▶ Network Appliance
<http://www.netapp.com>
- ▶ Blue Coat
<http://www.bluecoat.com>
- ▶ Cisco Cache Engine
<http://www.cisco.com>
- ▶ Edge Side Include (ESI)
<http://www.esi.org>
- ▶ Sun Solaris performance information
<http://www.sean.de/Solaris/soltune.html>
- ▶ IBM Rational Suite® TestStudio®
<http://www.ibm.com/software/awdtools/suite/>
- ▶ Rational Performance Tester
<http://www.ibm.com/software/awdtools/tester/performance/>

- ▶ ApacheBench manual and list of options
<http://httpd.apache.org/docs/programs/ab.html>
- ▶ Microsoft Data Access Components (MDAC) version 2.5 download
<http://microsoft.com/data/download.htm>
- ▶ OpenSTA V1.4.2
<http://www.opensta.org/download.html>
- ▶ OpenSTA community site
<http://portal.opensta.org/>
- ▶ OpenSTA sourcecode
<http://opensta.sourceforge.net/>
- ▶ JMeter, Open Source software from the Apache Software Foundation
<http://jakarta.apache.org/jmeter/>
- ▶ TestMaker and TestNetwork, from PushToTest
<http://www.pushtotest.com/>
- ▶ Grinder
<http://grinder.sourceforge.net>
- ▶ LoadRunner from Mercury Interactive
<http://www.mercury.com/us/products/performance-center/loadrunner/>
- ▶ Segue SilkPerformer
<http://www.segue.com/products/load-stress-performance-testing/>
- ▶ WebLOAD from Radview
<http://www.radview.com/products/WebLOAD.asp>
- ▶ WebStone from Mindcraft
<http://www.mindcraft.com/webstone/>
- ▶ OpenLoad from Opendemand Systems
<http://www.opendemand.com/openload/>
- ▶ Implementing a Highly Available Infrastructure for WebSphere Application Server Network Deployment, Version 5.0 without Clustering
http://www.ibm.com/developerworks/websphere/library/techarticles/0304_alcott/alcott.html

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Archived

Index

Numerics

2PC transaction 485
2phase commit 485, 666, 714
3DES 1022
80/20 rule 898

A

Abstract Parent Class 908
Access intent 928, 991, 996–997
 Read 995
 Settings 32
Access log 1019
Access patterns 982
Access plan 1027
action handler 903
ActionServlet 538, 901
Activate at 992
 Once 992
 Transaction 992
Activation specification 647, 652, 670
ActiveCount 988
activity.log 304, 307
ActivitySession 998
Adapter Bean Name 566
admin_host 232
Administrative cell 14
Administrative Console 13
 Manage Web servers 244
 Propagate plug-in configuration file 244
 Regenerate plug-in 260
Administrative service 13, 773
ADV_was.java 193–194
Affinity
 Process 357, 365, 370
 Transaction 345, 357, 365
Agent 847
Agent Controller 844–847, 855
 Agent 847
 Client 847
 Host process 846
 J2EE Request Profiler 847
 Java Profiling Agent 847
AIO device 206
Akamai 504, 566
Akamai EdgeSuite 566
Alarm Manager 779
Alexandre Polozoff 941
Apache 13, 1015
Apache HTTP Server 945
Apache JMeter 856, 897
Apache Software Foundation 901
Appliance server 53, 58
Application
 80/20 rule 898
 Architecture 897, 930
 Caching technologies 897
 Design 62, 897, 930
 Execution flow 852
 Life cycle 897
 Performance 896, 930
 Performance bottleneck 897
 Performance optimization 897
 Scalable 897
Application assembly performance 991
Application clients 72
Application design 40
Application development
 Coding issues 936
 Form submission 901
 High-volume Web sites 899
 JavaServer Pages 899
 Memory allocation 930
 Memory deallocation 930
 Memory usage 930
 Performance target 897
 Performance testing 840
 Reclaim memory 930
 Servlets 899
Application maintenance 919
Application management 829
 Enterprise view 829
 Infrastructure testing 832
 Measure end-to-end response time 832
 Monitoring
 Account and Server Group Management 836
 Active 832

- Client transactions 832
- Memory Analysis 836
- Monitoring On Demand 836
- Passive 832
- Software Consistency Check 836
- System Resources 836
- Trap and Alert Management 836
- Performance management 830
- Problem and resolution process 830
- Problem determination 830
- Request decomposition 832
- Application performance 62, 896, 930
 - Health 1005
 - Testing 940
- Application profiling 840, 928, 997
 - Closed-world analysis 998
 - Open-world analysis 998
 - Performance characteristics 873
 - Profile 998
 - Task 998
 - Verify runtime behavior 873
- Application Response Measurement 807
- Application Server Node 83
- Application Server Samples
 - BeenThere 426
- Application Server Toolkit 195, 515, 586, 709, 998
 - Analysis engine 998
- ARM 771, 807
- ARM agent 807
- ARM Agents 805
- ARM API 807
- ARM interface 806
- Array 937
- AST *See* Application Server Toolkit
- Asynchronous messaging system 625, 650
- Automatic lock recovery 490
- Auto-reload
 - Web module 263
- Availability 4, 7, 100
 - EJB container 89
 - Failover 8
 - Hardware-based high availability 7
 - Maintainability 9
 - Web container 89
 - Web server 89
- Availability matrix 484
- Average concurrent users 940
- Average request rate 940
- Average servlet response time 805

Avg Method RT 775–776

B

- Back end application 627
- Backing bean 904
- Backout queues 728
- Backout threshold 728
- BackupServers tag 276
- Batch processing 54
- Batch size 636
- BEA Systems 924
- Bean cache 991
- Bean Managed Persistence *See* BMP
- Bean managed transactions *See* BMT
- Bean scope
 - application 900
 - page 900
 - request 900
 - session 900
- Beans.instantiate() 900
- BeenThere 309–310, 355, 426, 691
 - Application Server Samples 426
 - Download 426
 - Installation 427
- Benchmark 41
 - Trade 6 391, 436
- Benchmark Center 60
- Benchmarking 59
- bind 1027–1028
- BLOB 910
- Blue Coat 504
- BMC 63
- BMC Patrol 837
- BMP 925, 995
 - Entity beans 667, 925
 - Advantages 926
 - Disadvantages 925
- BMT 383
- Bootstrap 347
- Bootstrap server 347
- Borland Optimizelt 933
- Bottleneck 978, 980–981, 987–988, 990, 997, 1004
 - Processing 978
- Boundary statistic 773
- Bounded range statistic 774
- Broker 715
 - Administrator perspective 744
- browse-parse-get loop 638

- Buffer pool 1027
- Build Verification Test process 64
- Bulk encryption/decryption 1021
- Bus
 - Clustering support 656
 - Destinations 647
 - Members 645–646
 - Messaging engine
 - High availability 656
 - Workload management 657
 - Multiple 646
- Business Delegate 916
- Business Delegate pattern 915, 920
- Business logic layer 906, 912, 914, 916
 - Caching 919
- Business objects 912
- Business process model 912
- Business rules 912
- Business to business 48

C

- CA *See* Certificate Authority
- Cache 932, 985, 993
 - Discards 985
 - Flushing 937
- Cache - Control 505
- Cache appliances 504
- Cache consistency 548
- Cache ID 281, 541, 551, 568, 572
- Cache instance 509
 - JNDI name 512
 - Object 510
 - Servlet 510
- Cache invalidation 556
 - Event-based 556
 - Policy 556
 - Techniques 556
 - Time-based 556
- Cache monitor 546, 557
- Cache policy 570, 572, 907
- Cache replication 511, 524, 547
 - cachespec.xml configuration 553
 - Configuration
 - Push frequency 553
 - Runtime mode
 - Not Shared 552
 - Push and Pull 550
 - Push only 551

- Sharing policy 553
 - Not-shared 553
 - Shared push-pull 553
 - Shared-push 553
- Enabling 548
- Testing 554
- Troubleshooting 558
- Cache size 985–986
- CacheMonitor.ear 516
- CacheQueries 581
- cachespec.xml 511, 524, 526, 532, 542, 548, 556–557
 - Configuration 532, 553
 - Location 512
- cachespec.xml elements 512–514
 - command 512
 - EdgeCacheable 514
 - ExternalCache 514
 - JAXRPCClient 513
 - not-shared 513
 - servlet 512
 - shared-pull 513
 - shared-push 513
 - shared-push-pull 513
 - static 512
 - sub-elements 514
 - webservice 512
- CacheTimeMargin 581
- Caching 55, 58, 826, 1020
 - Business logic 919
 - Commands 502, 504
 - Content type
 - GIF 502
 - HTML 502
 - JPG 502
 - JSP rendering less dynamic data 502
 - Customized content 503
 - Database 505
 - DB2 DBCache 505
 - Oracle 9i IAS - Relational Cache 505
 - TimesTen 505
 - Versant enJin 505
 - Dynamic content 501, 503, 556
 - Fragments 506–507
 - In memory 568, 610
 - Invalidation 501
 - JSPs 504
 - Page fragments 504

- Personalized data 501
- Servlets 504, 826
- Static content 501–502, 556, 610
- To disk 610
- Web page design 507
- Web services 504, 826
- WebSphere commands 826
- Caching framework 502
- Caching Proxy 87, 122, 394, 504, 507, 566
 - Adapter module 578
 - AIX
 - Installation
 - Using SMIT 210
 - AIX Installation
 - bos.iocp.rte 205
 - IOCP device 205
 - Prerequisites 205
 - Cachable content 122
 - Cache store 219
 - Collocation with Web server 214
 - Configuration 212
 - ibmproxy.conf 218
 - Port 213
 - Reverse proxy 213
 - Configuration wizard 213
 - Configure dynamic caching 581–582
 - Data retrieval 122
 - Disk cache 219
 - Dynamic caching 124, 578
 - Cache synchronization 579
 - Forward proxy 122, 212
 - Installation
 - LaunchPad 206
 - Load balancing 124
 - Monitoring 582
 - Protocols 122
 - Reverse proxy 122–123
 - Start
 - AIX
 - startsrc 223
 - stopsrc 223
 - Windows Services 222
 - Statistics 571
 - Stop
 - Windows Services 222
 - Storage device 122
 - Target Web server 214
 - Test 224
 - Troubleshooting 584
 - Web administration tool 216
 - Caching strategy 66
 - Caching techniques
 - Caching Proxy 610
 - Dynamic caching 610
 - ESI 610
 - For static content 610
 - FRCA 610
 - Web server plug-in 610
 - Caching tiers 503
 - Casting 936
 - CBR *See* Content Based Routing
 - CDN *See* Content Delivery Network
 - Cell
 - Web server 245
 - Certificate Authority 244
 - CF *See* Connection factory
 - Charts
 - Tivoli Performance Viewer 802
 - CICS 777, 834–835
 - CIFS 490
 - Cipher suite 1022
 - CIS 54
 - Cisco Cache Engine 504
 - Cisco CSS Controller 110
 - Class
 - Garbage collection 1012
 - Instance of 862
 - Instance Statistics 859
 - Structure 936
 - Client 847
 - Client authentication 244
 - Client ID 716
 - CloneID 281
 - Cluster 19, 52, 228, 244, 343, 389, 990
 - Creation 403–404
 - Horizontal scaling 82
 - Vertical scaling 82
 - Cluster member 19, 235, 244
 - Configuring transports 235
 - Failover operation 320
 - Failure 321
 - Failure during request 331
 - Failure with active sessions 322
 - Marking down 274
 - Overload 334
 - Security 23
 - Stopping 321
 - Cluster routing table 359

- Cluster settings 491
- ClusterAddress tag 268, 271
- Clustered TM Policy 476, 489, 497
- Clustering software 496
- CMP 925, 995–996
 - Entity beans 666, 925, 995–996
 - Advantages 925
 - Disadvantages 926
- CMT 383
- Coding best practices 896, 930
- Cold failover 487
- Collection size 931
- com.ibm.CORBA.LocateRequestTimeout 375
- com.ibm.CORBA.MaxOpenConnections 1025
- com.ibm.CORBA.RequestRetriesCount 375
- com.ibm.CORBA.RequestRetriesDelay 375
- com.ibm.CORBA.RequestTimeout 375–376
- com.ibm.CORBA.ServerSocketQueueDepth 1024
- COMM_FAILURE 375
- Command bean 917
- Command Cache 540–541
- Command caching 34, 540, 609
 - Enabling 541
 - Testing 546
- Command Design Pattern 541
- Commits 1028
- Concurrency test
 - Memory leak 1006
- Concurrent requests 780
- Concurrent user 980
- Concurrent Waiters 983
- Conditional check 936
- Configuration repository 14
- Connection 652
 - Keep-alive 1022
- Connection backlog 334
- Connection factory 650, 652, 699, 701, 716
- Connection manager 977
- Connection pool 792, 935, 983–985
 - Maximum connections 633
- Connection pool size 983
- Connection pooling 55, 813, 983
 - Performance data 777
- ConnectionConsumer object 704
- Connections
 - Idle 722
- ConnectionWaitTimeout exception 667–668
- ConnectTimeout 335
- Container Managed Persistence *See* CMP

- Container Managed Transactions *See* CMT
- Container transactions 991, 999
 - Bean Managed 999, 1001
 - Mandatory 999–1000
 - Never 999
 - NotSupported 999–1000
 - Required 999–1000
 - Requires New 999–1000
 - Supports 999–1000
- Content Based Routing 109
- Content caching tiers
 - Client caching services 504
 - Dynamic cache service 504
 - External caching services 504
 - Internet content delivery services 504
- Content Delivery Network 504, 508
- Controller 907–908
- Controller servlets 908
- Cookie 909
- Cookies 282, 312, 616, 911
 - JSESSIONID 281
- CORBA 950
- CORBA CosNaming 372
- corbaloc provider URL 351
- Core group 358, 467
 - Bridge service 468
- Coordinator 468
 - Election 470
 - Failure 471
 - Preferred 469
- Member 467
- Member discovery 482
- Member status 482
 - Connected 482
 - In a view 482
 - Not connected 482
- Policy 467, 474, 660
 - Clustered TM Policy 489
- Configuration
 - Fail back 476
 - Preferred servers 476
 - Preferred servers only 476
 - Quorum 476
- Fail back 661
- Large clusters 661
- Match criteria 660
- Preferred servers 660
- Preferred servers only 661
- Type 475, 660

- All active 475
- M of N 475
- No operation 476
- One of N 475, 660
- Static 477
- Transport type 478
 - Channel Framework 479
 - Multicast 478
 - Unicast 478
- Correlation ID 638
- Cost/benefit analysis 56
- Count statistic 773, 780
- Counters 776, 792
 - Avg Method RT 775–776
 - Performance data 774
 - Selecting multiple 800
- Covalent Enterprise Ready Server 13
- CPU 614, 974
- CPU starving 484
- CPU utilization 63, 778, 971
- Create custom profile 400
- create method 355
- Create server cluster 403
- createSession 701
- Current weight 270
- Custom tag libraries *See* JSTL
- Customer Information Service 54
- Customer self-service 46
- CustomLog 1019

D

- Data
 - Caching 937
- Data access frameworks 920
- Data access layer 919
- Data access logic 926
- Data counters 779
- Data Mediator Services 922–923
 - EJB 922
 - JDBC 922
- Data mediators 920
- Data Replication Service *See* DRS
- Data server 42
- Data source 921
- Data source queues 983
- Data Transfer Object pattern 915, 920
- Data Transfer Objects 916, 923
 - XML 906

- Data transmission time 1016
- Database connection 935
- Database connection pools 792
- Database indexes 57
- Database manager overhead 1028
- Database persistence 280
- Database row lock 995
- Database session management
 - Session persistence 291
- Datasource 935
- DB2 87, 291, 438, 676, 734, 742, 1026
 - Bufpage 1027
 - catalog command 439
 - Configuration 438
 - Logging 1026
 - MaxAgents 1026
 - MaxAppls 1026
 - MinCommit 1028
 - Multi-row feature 295
 - Performance
 - Repeatable read 994
 - Query optimization level 1027
 - reorgchk 1028
 - Transaction log 1028
 - Variable row size 295
- DB2 configuration advisor 1026
- DB2 DBCache 505
- DB2 Information Integrator 54
- DB2 Legacy CLI-based Type 2 JDBC driver 676
- DB2 UDB 835
- DB2 universal JDBC Type 4 driver 439, 676, 750
- db2cmd 438
- DB2COMM 1026
- DBMS connectivity 919
- DCP 509
- DCS 32, 466, 472, 474, 483
 - Heartbeat 483
- Dead Letter Queue 738
- Deadlock 974, 984
- Deadlock situation 629
- Default messaging provider 13, 34, 628, 643, 713
 - Activation specification 652, 670
 - Activation specification settings
 - Maximum batch size 670
 - Maximum concurrent endpoints 670
 - Add server cluster to bus 646
 - Add server to bus 646
 - Bus members 646
 - CF settings

- Connection proximity 664
 - Bus 664
 - Cluster 664
 - Host 665
 - Server 665
- Nonpersistent message 665
- Persistent message 665
- Read ahead 665
- Share datasource with CMP 666
- Target 663
- Target significance
 - Preferred 664
 - Required 664
- Target type 663
- Clustered messaging engines 659
- Clustering 656
- Component relationships 655
- Components
 - Connection factory 652
 - Destination 650
 - Message queue 650
 - Target destination 652
- Configuration guidelines 663
- Connection pooling 631
- Connection pools settings
 - Aged timeout 670
 - Configuration 667
 - Connection timeout 667
 - Maximum connections 668
 - Minimum connections 669
 - Purge policy 670
 - Reap time 669
 - Unused timeout 669
- Destinations 647
 - Alias 647
 - Foreign 647
 - Queue 647
 - Queue point 647
 - Topic space 647
- High availability 656
- JCA
 - Resource Adapter 652
- JCA connection pool 655
- Message destination 650
- Message reliability 648
 - Performance 648
- Messages
 - Levels of reliability 648
- Messaging destination
 - Exception destination 671
 - Maximum failed deliveries 671
- Messaging engine 646
 - Data store 649
 - Messaging engine name 646
 - Partitioned queues 657
 - Local partitions 657
 - Remote partition 658
 - Performance 643, 657
 - Preferred servers 660
 - Quality of Service 649
 - Workload management 656
- Default SIBus Policy 476
- default_host 232
- DefaultApplication.ear 309
- DefaultCoreGroup 476
- DeliveryMode.NON_PERSISTENT 712
- DeliveryMode.PERSISTENT 712
- Dependency ID 542, 544
- Dependent value classes 927
- Deployment descriptor 348, 915, 925, 928, 998
- Deployment Manager 7, 12, 87
 - Failure 374
 - plugin-cfg.xml 260
 - Profile 396
 - SPOF 358
 - Windows Service 400
- DES 1022
- Design
 - Application 40
 - Implementation 41
- Destination 650, 699
- destroy() method 901
- Development cycle 840
- Development environment 231, 263
- Development life cycle 907
- Dirty reads 994
- Disk access 56
- Disk IO 974
- Disk utilization 63
- Dispatcher 102
 - Advisors 104, 107
 - Connect 108
 - connecttimeout 108
 - Custom 109
 - Customizable 108
 - Downed server 108
 - receivetimeout 108
 - Executor 103

- Connection table 103
 - Forwarding methods 105
 - CBR 107
 - MAC forwarding 105
 - NAT/NAPT 106
 - Return address 107
 - Manager 103
 - Metric Server 104
 - Server weights 102
 - Dynamic 102
 - Fixed 102
 - Distributable 991, 1001
 - Distributed Fragment Caching and Assembly Support 566
 - DistributedMap 510
 - DistributedMap caching 34
 - DistributedObjectCache 510
 - Distribution and Consistency Services *See* DCS
 - DMap caching *See* DistributedMap caching
 - DMS 922
 - doGet method 651, 700, 900
 - Domain Replication Service 32
 - doPost method 900
 - DRS 26, 78, 294, 297–298, 377, 472
 - Number of replicas 299
 - Replication domain 299
 - Topologies 299
 - dscontrol 136, 201
 - cluster configure 162
 - executor report 157
 - executor stop 204
 - high status 178
 - high takeover 180
 - manager report 156, 191
 - dsserver 201, 203–204
 - Start Dispatcher 136
 - stop 204
 - DTO 916, 923
 - Durable subscription 715, 717
 - ID 716
 - DynaCacheEsi application 523
 - dynaedge-cfg.xml 580
 - Dynamic Cache Monitor 516, 523, 571, 826
 - Edge Statistics 571
 - Installation 517–518, 520–521, 561, 563
 - Dynamic Cache Policy Editor 515
 - Installation 586
 - Dynamic cache service 392, 502, 778, 826, 899, 938, 1020
 - Cache replication 524
 - Push frequency 550
 - Command caching 508
 - Configuration 515, 524
 - Disk off-load
 - LRU algorithm 526
 - Priority weighting 526
 - Disk offload 524, 526
 - Enable 526
 - Tuning 526
 - DistributedMap 34, 510
 - DistributedObjectCache 510
 - Enabling 524
 - JSP result caching 507
 - Servlet caching 507
 - Troubleshooting 557
 - Dynamic caching 907, 919
 - Cache consistency 548
 - Cache instance 509
 - Cache replication 511, 547
 - Caching Proxy adapter module 578
 - Command Design Pattern 541
 - DistributedMap 437
 - Dynamic Content Provider 509
 - Edge Side Includes 514
 - External cache group name 514
 - Struts and Tiles caching 537
 - WebSphere Command Framework API 541
 - Dynamic content 503
 - Dynamic Content Provider 509
 - Dynamic SQL statements 1027
- ## E
- Eclipse 844, 846, 956
 - Eclipse Hyades 844
 - Edge Components 53, 84, 87, 99, 101, 388, 394, 504, 571
 - Caching Proxy 122, 507–508
 - LaunchPad 125
 - New features in V6 125
 - Supported platforms 101
 - Edge of Network Caching Support 609
 - Edge server 42, 58
 - Edge Side Includes *See* ESI
 - Edge Statistics 571
 - EIS *See* Enterprise Information System
 - EJB 23–24, 883, 912–913, 986, 995
 - Bootstrapping 347

- Caching options 29, 345, 927
 - Option A caching 345, 383, 992–993
 - Option B caching 345, 383, 992–993
 - Option C caching 346, 383, 992–993
- Client 343, 927, 986
- Cluster 407
- Deployment descriptor 639, 915
- Entity bean 28, 345
- Entity bean home 343
- Entity bean instance 343
- Entity beans 926
 - Alternatives 928
 - Coding guidelines 926
 - Design guidelines 926
- High availability 343
- Home 349, 355, 914
- Inheritance 927
- InitialContext 914
- Isolation levels 928, 995
- JNDI lookup
 - In same cell 347
 - Outside cell 351
 - WebSphere client container 351
- Load balancing 343
- Local interface 914, 928
- Lookup operation 914
- Object reference 344
- Passivation 346
- read-only method 995
- Reference 348–349, 914
- Reference mapping 349
- Remote method call 914
- Response time 792
- Server affinity 30
- Session 28
- Session bean home 343
- Stateful session beans 28, 344
 - Failover 344
- Stateless session bean instance 343
- Stateless session beans 28, 344
- Transaction 28
- WLM 341
- Workload management policy 357
- EJB 2.0 914, 927, 996, 1001
- EJB caching
 - Option A caching 345, 383, 992–993
 - Option B caching 345, 383, 992–993
 - Option C caching 346, 383, 992–993
- EJB Command Framework 917
- Disadvantages 918
- EJB container 347, 366, 389, 913, 917, 925, 927, 986
 - Availability 89
 - Cache settings 986
 - Cache size 986
 - Cleanup interval 986
 - Failover
 - Tuning 375
 - Pool size 635
 - Thread pool 627
 - Transaction management 915
 - Workload management 389
- EJB server selection policy
 - Prefer local 358, 366
 - Process affinity 358, 360, 368, 370
 - Server weighted round robin routing 358–359
 - Transaction affinity 358–359, 368, 370
- EJB session bean 908, 926, 929
 - Direct access to back end 929
 - Advantages 929
 - Alternatives 930
 - Disadvantages 929
- EJB stubs 917
- EJB WLM 343
 - Cluster configuration changes 357
 - Configuration changes 357
 - Fairness balancing 362
 - Initial request 355
 - Prefer local 358, 366
 - Configuration 367
 - Runtime changes 368
 - Process affinity 357–358, 360, 365, 368, 370
 - Selection policy 357
 - Server weighted round robin
 - Configuration 362
 - Server weighted round robin routing 358–359
 - Configuration
 - Runtime changes 363
 - Transaction affinity 345, 357–359, 365, 368, 370
- EJB workload management *See* EJB WLM
- ejbActivate 993
- ejbLoad() 925
- EJBLocalObject 383
- EJBObject 383
- ejbStore() 925
- EJBTM 437

- EJS WLM 23
- EJS workload management 17
 - Enabling 342
 - Entity bean 345
 - Failover 371
 - Normal operation 354
 - Server selection policy 358
 - Stateful session beans 344
 - Stateless session beans 344
- Embedded HTTP transport 13, 73
- en0 141
- en1 141
- Enable clone support 715
- Enterprise beans
 - Performance data 777
- Enterprise Information System 919–920
- Enterprise JavaBean *See* EJB
- Enterprise view of applications 829
- Entity bean 28, 345, 919, 924, 984
 - Failover 383
 - Server affinity 30
- Entity EJBs 924
 - Activate at 992
 - Load at 992
 - Manipulation of data 927
 - Result sets 927
- Environment
 - Development 231, 263
 - Production 231
- Error condition 937
- error.log 308
- error_log 308
- ESI 504, 508, 559, 566
 - Best practices 615
 - Cache 568
 - Invalidation 574
 - Cache statistics 571
 - Cache Hits 572
 - Cache Misses By Cache ID 572
 - Cache Misses By URL 572
 - Cache Time Outs 572
 - Content 573
 - ESI Processes 572
 - Evictions 573
 - Number of Edge Cached Entries 572
 - include tags 567–568
 - Performance considerations 614
 - Processor 523, 560, 568, 610
 - Configuration 569
 - Processor cache 571
 - Request 572
 - Surrogate capability 508
- ESI/1.0 505
- esiInvalidationMonitor 523, 569
- Event Broker
 - Execution Group 745
 - Message flow 745
- Event Integration 833
- Exception 937, 974, 1028
 - Catching 937
 - Throwing 937
- execute() 1020
- Execution flow 869
- Execution Flow view 867
- Execution Group 744
- Execution time
 - Profiler
 - Execution time 858
- Extensible Stylesheet Language Transformation *See* XSLT
- External cache
 - Invalidation 585
- External Cache Adapter 524
- External caching scenarios 558
 - Configuration
 - Adapter Bean Name 566
 - AFPA 564
 - Caching Proxy 565
 - ESI 568
 - EsInvalidator 565–566
 - External cache group 560
 - Cache group members 565
 - Web server plug-in 568
 - IBM HTTP Server's high-speed cache 559
 - Using Caching Proxy 578
 - Configuration
 - External cache group 579
 - Using Edge Components 559
 - Using ESI 558–559, 566
 - Using FRCA 559, 574
 - Configuration
 - External cache group 575
 - Using Web server plug-in 558, 566
 - External clustering software 476

F

- Facade 370, 915, 927

- Message 914
- Session 913
- Failover 8, 16, 21, 228, 244
 - Cold 487
 - Hot 489
 - ORB plug-in 371
 - Primary and backup servers 275
 - Stateful session beans 377
 - Best practices 383
 - Configuration 378
 - Application level 379
 - EJB container level 378
 - EJB module level 381
 - Stateless session beans 377
 - Tuning 335
 - Web server plug-in 275
- Failover tuning
 - ConnectTimeout 335
 - RetryInterval 337
- Failure
 - Deployment Manager 374
- Fast Response Cache Accelerator *See* FRCA
- Fat client 908
- Fault isolation 10
- Federated node 400
- Field
 - Static 933
- File system locking 490
- filemon 1013
- Filter 812
- FIN_WAIT 483
- Final modifier 937
- finally block 641
- findByPrimaryKey 995
- Firewall 11
- Fragment caching 506–507
- Fragmentation 930
- FRCA 508, 559, 564, 566, 574, 577
 - Monitoring cache 578
- Functional Verification Test 64
- FVT 64

G

- Garbage collection 469, 786–787, 837, 863, 910, 931, 936, 980, 1008
 - Bottleneck 1004–1005
 - Class 1012
 - Common problems 1010

- Compaction phase 1003
- Concurrent mark mode 1004
- Incremental compaction mode 1004
- Mark phase 1003
- Monitoring 1005
- Parallel mark mode 1004
- Parallel sweep mode 1004
- Phases 1003
- Sweep phase 1003
- Time 933
- Garbage collection call 933, 1005
 - Duration of 1008
 - Length of 1009
 - Number of 1008
 - Time between calls 1009
- Garbage Collector 930, 1002
 - Mark - Sweep - Compact technique 1002
- GC *See* Garbage collection
- Generic JVM arguments 787
- Generic messaging provider 628
- genericJvmArguments 790
- GenPluginCfg 260–261
 - wsadmin 260–261
- get-by-correlationId 638
- get-by-messageId 638
- getConnection() 668
- getIntValue() 913
- getLongValue() 913
- getRuntimeEnvInfo method 355
- getter type method 995
- Google 964
- Grinder 965

H

- HA group 474
- HA policy 358
- HA software 489, 713
- HACMP 465, 468, 476, 488, 713, 734
- HAManager 32, 297, 358, 374, 466, 660
 - Core group 467
 - Member 467
 - Policy 467
 - Core group bridge service 468
 - Core group policy 474
 - Failure detection 483
 - Active 483
 - TCP KEEP_ALIVE 485
 - Failure detection time 483

- High availability group 479
- Match criteria 477
- Messaging services 466
- Policy 497
- Server failure 466
- Singleton service 466
- Transaction service HA 466
- Transport buffer 472
- View 482
- View installation 483
- HAManager MBean 476
- Handshake 1021
- Harden Get Backout 729
- Hardened 729
- Hardware
 - Requirements 388
- Hardware accelerator 1022
- Hardware-based high availability 7
- Hashtable 1005, 1007
- Heap 722, 787, 975, 991
 - Allocations 786
 - Compaction 933, 1003
 - Consumption 1007
 - Expansion 1003
 - Fragmentation 1007
 - Number of objects in heap 933
 - Parameters 1002, 1008
 - Shrinkage 1003
 - Size 813, 933, 1005, 1011
 - Initial 1008, 1011
 - Maximum 1005, 1008, 1011
 - Minimum 1005
 - Tuning 1009
 - Space 57
 - Thrashing 1009
 - Xgcpolicy 1004
 - Xmaxe 1003
 - Xmaxf 1003
 - Xmine 1003
 - Xminf 1003
 - Xms 1003
 - Xmx 1003
- Hidden form fields 911
- High availability
 - EJB bootstrap failover 371
 - EJB container redundancy 372
- High availability group 467, 474, 479
- High Availability Manager *See* HAManager
- High availability software 85
- High-volume Web sites
 - Application development 899
- HitCount 309, 313
- Hits per
 - Day 940
 - Hour 940
 - Minute 940
 - Month 940
- Home 343, 355
- Horizontal scaling 22, 82, 84
- Host alias 516
- Host Process 846
- Hosts interactions 869
- Hot failover 489
- HP-UX 11i 1014
- htcformat 219–220, 584
- HTML frames 911
- HTTP 229, 908, 942
- HTTP advisor 93
- HTTP cache 505
- HTTP cache directives 504
- HTTP channel 235
- HTTP GET 945, 953
- HTTP Inbound Channel 989
- HTTP POST 945
- HTTP requests 937
- HTTP response code 503 338–339
- HTTP server *See* Web server
- HTTP session 25, 383, 909
 - Invalidation 910
 - Server affinity 30
 - Size 910
 - Timeout 11
- HTTP transport 13, 234, 239
 - Embedded 73
- HTTP transport channel
 - Maximum persistent requests 988
 - Read timeout 989
- HTTP transport port 516
- HTTP tunneling 479
- HTTP/1.1 505
- HTTP/1.1 caching directives 505
- http_plugin.log 304, 809
- httpd command 1017
- httpd.conf 826, 1019
- HTTPS 229, 479, 908
- HTTPS transport 239
- HttpServlet.init() method 900
- HttpServlet.service() 303

HttpSessionBindingListener 911
HttpSessions 900
Hyades 844

I

IBM AIX 5.2 388
IBM alphaWorks 840, 886
IBM Business Workload Manager 833
IBM DB2 UDB 926
IBM DB2 UDB 8.2 676, 689
IBM DB2 UDB V8.1 388
IBM eServer Benchmark Centers 60
IBM Global Services 61
IBM HTTP Server 12, 80, 87, 251, 504, 609, 826, 945, 1015
 Admin Process 80
 Error log 308
 mod_deflate 1016
 Monitoring 826
 Multi-processing 1017
 Server-status page 826
 Tuning
 MaxClients 1018
 MaxRequestsPerChild 1018
 MaxSpareThreads 1018
 MinSpareThreads 1018
 ServerLimit 1018
 StartServers 1018
 ThreadLimit 1018
 ThreadsPerChild 1018
 Use as external cache 574
IBM HTTP Server 2.0.42.2 388
IBM HTTP Server powered by Apache 945
IBM JDK 1.4.1 388
IBM ORB 355
IBM Rational Agent Controller 844–846, 874
IBM Rational Application Developer *See* Rational Application Developer
IBM Rational Performance Tester 956–957
 HTTP Proxy Recorder 959
 Performance Schedule
 Create 961
 Create Loop 962
 Run 962
 Performance Summary Report 962
 Performance Test
 Record 958
 Reports 963

 Response vs Time 963
 Test Contents 961
 Virtual test users 962
IBM Rational Software Development Platform 958
IBM service log file 828
IBM Test Center 60
IBM Tivoli Monitoring for Transaction Performance 803
IBM Tivoli Monitoring for Web Infrastructure 803
IBM Tivoli performance monitoring tools 790
IBM TotalStorage 490
IBM WebSphere Application Server Network Deployment V5.1 388
ibmproxy.conf 581, 584
IBMSession 304
Identifying field 639
ifconfig 155
if-then 936
IGS 61
IHS 80
Implementation design 41
IMS 777, 834–835
Infrastructure testing 832
init() 1002
Initial context 624
Initial heap size 1008
InitialContext 347, 352, 372, 375, 914
Initialization
 Lazy 931
initialization routines 1002
In-memory cache 568
In-process request 370
Installation
 Application 427
 Instance 343, 774
 Instance Statistics 862
Instrumentation level
 Setting 780
Interface
 Implementation 913
 local 927
 Remote 917
Internet Explorer 960
Internet Information Server 13
Internet Services Provider 504
Interoperable Object Reference *See* IOR
Invalidation 553, 556
 ESI cache 574
 Event-based 556

- Event-driven 559
- External cache 585
- Policy 556
- Techniques 556
- Time-based 556
- Invalidation API 556
- IOR 356
 - Direct 356
 - Indirect 356
- iostat 1029
- IP sprayer *See* Load Balancer
- IP spraying 100
- IP-forwarding 123
- iPlanet 1019
- Isolation levels 991, 993, 995, 997
 - Database overhead 994
 - Row locking 994
- ISP *See* Internet Services Providers
- ITSO sample topology 86, 388, 403

J

- J2C Authentication data 750
- J2C connectors
 - Performance data 777
- J2EE 72–73, 341, 834
 - APIs 73
 - Application 831, 1020
 - Application client container 351
 - Applications 621
 - Business logic tier 73
 - Client application
 - Prefer local policy 366
 - Client tier 72
 - Components 228, 883
 - Context lookup 624
 - Context reference 624
 - Enterprise Information System tier 73
 - Management specifications 773
 - Multi-tier environment 72
 - Name 716
 - Naming 347
 - Presentation tier 72
 - Request Profiler 847, 875
 - Request Profiler Agent 867
 - Security 909
 - Specification 900
 - Tiers model 72
- J2EE 1.3 437, 625

- J2EE 1.4 437, 621, 914
- J2EE 1.4 compliant messaging provider 621
- J2EE Connector Architecture *See* JCA
- J2EE Specification 924
- J2EETM) 437
- Jakarta project 901
- Java
 - Access to synchronized resources 934
 - Buffered I/O classes 937
 - Casting 936
 - Class structure 936
 - Client 16, 343, 360, 913
 - Bootstrapping 352
 - Conditional check 936
 - Copying contents of arrays 937
 - Externalization 910
 - Final modifier 937
 - I/O libraries 934
 - if-then 936
 - Memory management 931
 - Memory tuning 1002
 - Monitors 934
 - Performance
 - Swapping 1011
 - Process ID 359
 - Reflection 936
 - Strings 936
 - Synchronization 933
 - Using ? for conditional check 936
- Java 2 Platform Enterprise Edition *See* J2EE
- Java Community Process 928
- Java Connector Architecture *See* JCA
- Java Data Objects *See* JDO
- Java Database Connectivity *See* JDBC
- Java Management Extension 772, 804
- Java Message Service Server *See* JMS
- Java Messaging Service *See* JMS
- Java Naming and Directory Interface *See* JNDI
- Java process
 - Attach to 849
- Java Profiling Agent 847, 875–876, 1005
- Java Specification Request 924
- Java Virtual Machine 75, 370, 777, 927
- Java Virtual Machine Profiler Interface *See* JVMPi
- java.lang.String 936
- java.lang.StringBuffer 936
- java.rmi.RemoteException 999
- java.sql.Connection 935
- java.sql.PreparedStatement 936

- java.sql.ResultSet 935
- java.sql.Statement 935–936
- java.util.ArrayList 934
- java.util.Hashtable 931, 934
- java.util.Vector 931, 934
- java: lookup names 352
- JavaBeans 912
- JavaServer Faces *See* JSF
- JavaServer Pages *See* JSP
- javax.jms.MessageProducer 712
- javax.jts.TransactionRequiredException 1000
- javax.servlet.Servlet.init() method 900
- javax.transaction.UserTransaction 1000
- JAX-RPC 437
- JCA 631, 647–648, 777, 924, 933
 - Adapter 926
 - Connection pool 655
- JDBC 641, 901, 919, 927, 933, 935
 - Cached Rowset 924
 - Driver 936
 - Provider 292, 984
 - Connection pool settings 984
 - Resources 676, 688
 - Rowset 924
- JDO 919, 924, 928
 - Advantages 929
 - Alternatives 929
 - Data access 928
 - Disadvantages 929
- JFS *See* Journaled File System
- Jinsight 981
- JIT compiler *See* Just in Time compiler
- JMeter 897, 964
- JMS 13, 485, 621, 625, 650, 908, 917, 933
 - Activation specification 647, 670
 - Maximum batch size 635
 - Maximum endpoints 635
 - Activation specification settings
 - Maximum batch size 670
 - Maximum concurrent endpoints 670
 - API 625, 650
 - Receive messages 625, 650
 - Send messages 625, 650
 - Average workload arrival rate 719
 - Class 712
 - Client subscription
 - Durable 715
 - Non-durable 715
 - Component settings 711
 - Components 643, 705
 - Connection factory 701
 - Destination 699
 - JNDI Name service 650, 699, 702
 - Listener port 702
 - Message destination 699
 - Queue 699
 - Topic 699
 - Message queue 702
 - Queue destination 702
 - Queue manager 652, 699, 702
 - Target destination 701
 - Configuration
 - Enable clone support 715
 - Enable XA support 714
 - Listener port maximum sessions 718
 - Maximum messages 720
 - Maximum session pool size 724
 - Maximum sessions 720
 - Message persistence 712
 - Transport Type
 - BINDINGS 712
 - CLIENT 712
 - Configuration guidelines 708
 - Connections 624, 652, 722
 - Correlation ID 638
 - Default messaging provider 643
 - Destination 652, 701, 726
 - Durable subscription 717
 - ID 716
 - Failure 724
 - High availability 485
 - Infrastructure 705
 - Listener port 702
 - Message destination 699
 - Message ID 638
 - Message listener service 702–703
 - Message persistence 641
 - MessageProducer 649
 - Objects 716
 - Peak workload arrival rate 719
 - Performance tuning 719
 - Point-to-point messaging 655
 - Provider 638
 - WebSphere MQ 704
 - Provider code 728
 - Publish/subscribe 655, 715
 - QCF 704
 - Queue 747

- Queue destination settings 713
- Queue reader thread 720
- Resource adapter 625, 635, 671
- Resources 676, 688
- Session 721–722
- Specification 638
- Topic destination settings 713
- Usage 625
- JMS 1.1 unified API 699
- JMX 771–772, 804
- JNDI 355, 624, 649, 935, 937
 - Lookup 347–348, 652, 654, 701, 704, 900
 - Server cluster 353
 - Fault-tolerant 354
 - Single server 352
 - Name 430, 710
 - Fully qualified 348
 - Name resolution 348
 - Name service 650, 652, 699, 701–702
 - Namespace 624, 651, 700
- Journalized File System 476, 496, 1013
- JProbe 981
- JSESSIONID 281–282, 297, 534
- JSF 899, 903, 908, 923
 - Backing bean 904
 - Data conversion 904
 - Internationalization support 904
 - JSP tag library 904
 - Page navigation 904
 - UI component
- JSP 911, 998
 - Compositional 899
- JSP 2.0 899
- jsp include tag 511, 899
- jsp usebean tag 900
- JSP/servlet caching 907
- JSTL 899, 901
- Just In Time compiler 1011
- JVM 75, 366, 370, 467, 504, 720, 777, 927
 - Memory utilization 837
- JVM log 557
- JVM resource 722
- JVM tuning 1011
- JVMPI 771, 777, 786, 846–847, 972, 1005
 - Disabling using Administrative Console 790
 - Disabling using wsadmin 790
 - Enabling using Administrative Console 787
 - Enabling using wsadmin 789
- JVMPI profiler 1005

K

- keep-alive connection 1022
- kill command 494

L

- Latency 43, 1020
- Layers
 - Business logic 912
 - Data access 919
 - Presentation 898
- Lazy initialization 931
- lbadm 136
- LDAP 1021
- Least recently used *See* LRU algorithm
- Life cycle 777
- Linux scheduler 1014
- Listener 707
 - Service thread pool 722
- Listener port 702, 716, 721
 - Configuration 704
 - Failure 726
 - Initialization 703
 - Maximum connection pool size 724
 - Maximum number of sessions 718
 - Maximum retries 728
 - Maximum sessions 633, 718
 - Minimum connection pool size 724
 - Retry interval 728
 - Retry settings 726
 - Session 704, 707
 - Sessions 721
- Load at 992
 - Activation 992
 - Transaction 992
- Load Balancer 84, 87, 91
 - Advisor logging level 202
 - Advisor status 201
 - Advisors 146
 - Connect 146
 - Custom 193
 - Customizable 190
 - HTTP 146
 - AIX Installation 132
 - CBR component 109
 - Cisco CSS Controller 110
 - Cluster IP alias 162
 - Command line interface
 - dscontrol 136

- Configuration 394
 - Add cluster 138
 - Add Web servers 143
 - Basic scenario 136
 - Client gateway address 181
 - Executor 137
 - Network router address 186
 - Port 141
 - Return address 185
 - Save 146
- Content Based Routing 109
- Custom advisors 91, 109
- Dispatcher 102
 - Automatic start 203
 - CBR forwarding method 109
 - Scripts
 - goActive 119
 - goldle 119
 - goInOp 119
 - goStandby 119
 - highavailChange 120
 - serverDown 120
 - serverUp 120
 - Start 136
 - Start automatically 203
- Executor
 - Start 137
- Forwarding method
 - NAT/NAPT 181
- Graphical user interface
 - ladmin 136
- High availability 116, 162
 - Active server 117
 - Backup server 117
 - Backup server configuration 168
 - Configuration 164
 - Dispatcher scripts 119
 - Failover 179
 - Primary server 117
 - Primary server configuration 162
 - Reach target 118
 - Recovery 117
 - Recovery strategy 164
 - Server role 164
 - Server state 162
 - Standby server 117
- High availability configuration
 - Reach target 169
 - Test 177
 - Test automatic recovery strategy 177
 - Test manual recovery strategy 180
- High availability scripts 162
 - Configuration 171
 - goActive 172
 - goInOp 173
 - goStandby 172
 - highavailChange 174
 - serverDown 174
 - serverUp 174
- HTTP Advisor Request 191
- Installation 128
- Installation and Configuration 128
- Installation wizard 128
- LaunchPad 128
- Log file 202
- Manager
 - Log 145
 - Start 144
- Mutual High availability 118
- Nortel Alteon Controller 110
- On dedicated server 114
- Protocols 102
- Server affinity 110
 - Active cookie affinity 113
 - Cross port affinity 112
 - Passive cookie affinity 113
 - SSL session ID 114
 - Stickyness to source IP address 111
 - URI affinity 113
- Server monitor 156
- Site Selector 110
- SPOF 116
- Stop all components 204
- Testing
 - NAT 189
- Topologies 114
 - Collocated 115
 - High availability 116
 - Mutual high availability 118
 - On dedicated server 114
- Windows
 - Dispatcher service 203
- Load Balancer Node 84
- Load balancing 8, 16, 20, 84
- Load factors 66
- Load information 778
- Load on startup 991, 1001
- Load testing tools 945

- Load tests 66
- Load value 777
- LoadRunner 965
- Local cache 504
- Location Service Daemon 356, 374
- Lock lease 490
- Lock recovery 490
- Log
 - System.out 808
 - Viewing with Tivoli Performance Viewer 793
- Log Analyzer 828
 - Symptom database 828
- Log4J 935
- Logging 934
 - Reduction of 935
- Logging library
 - Multithreaded 935
- LogLevel 305
- Logout functionality 910
- Long-running test
 - Memory leak 1006
- Lookup 935
- Loop 937
- Loop iteration 937
- Loopback interface
 - Alias 148
 - AIX 155
 - Other platforms 155
 - Windows 149
- Lotus Domino 13
- LRU algorithm 526
- LSD 373
- LSD *See* Location Service Daemon

M

- Maintainability 4, 9
 - Configuration
 - Dynamic 9
 - Mixed 9
 - Fault isolation 10
- Managed node 79, 400
- Managed node custom profile 396
- Managing state 24
- Mandatory 915, 999–1000
- Match criteria 477
- Max Application Concurrency 982
- Max Connections 983–984
- MAX.RECOVERY.RETRIES 726

- MaxClients 1018
- Maximum concurrency level 983
- Maximum concurrency point 982
- Maximum heap size 1005, 1008
- Maximum messages 719–720
- Maximum number of connections 337
- Maximum number of threads 334, 1018
- Maximum persistent requests 989
- Maximum sessions 720
- Maximum size 987
- Maximum thread pool size 725
- Maximum weight 269
- MaxRequestsPerChild 1018
- MaxSpareThreads 1018
- maxWeight 270
- MBeans 773
- MDB 29, 625, 632, 647, 650, 652–654, 671–672, 702, 704–705, 715, 728, 730, 913, 917, 937, 998
 - Listener ports 716–717
 - onMessage 653, 702, 718, 721
- Mediation 908
- Memory 777, 974
 - Allocation error 1013
 - Allocation fault 1002
 - Leak 834, 841, 901, 930, 932, 935, 969, 1002, 1005, 1009
 - Concurrency test 1006
 - Detection 837
 - Long-running test 1006
 - Repetitive test 1006
 - System instability 1005
 - System test 1006
 - Testing 1006–1007
 - Tivoli Performance Viewer 1007
 - Overuse 1005
 - Utilization 63, 1004
- Memory-to-memory replication 26, 78, 280
 - Partition ID 297
 - Replication domain 295
- Mercury LoadRunner 945
- Message
 - Backlog 719
 - Producer 647, 658
 - Queue 650, 702
 - Reliability 648
 - Selector 639
- Message broker 716, 728, 742
 - Connection timeout 728
- Message consumer 647, 658

- Message destination 699
 - Queue 699
 - Topic 699
- Message driven bean 647
- Message facade 914, 917
- Message ID 638
- Message listener
 - Service thread 634
 - Threads 707
- Message listener service 640, 702–703, 705, 726
 - Maximum sessions 721
 - Maximum threads 633
- Message listener threads 707, 721, 765
- Message persistence 641, 712
 - Configuration values 713
- Message retention
 - Disable 640
- MessageConsumer 641, 716
- Message-driven beans 29, 625, 632, 647, 650, 652–653, 702, 913, 917, 937
- Messaging
 - Acknowledge mode 636, 670
 - Bus 645
 - Connection 622
 - QueueConnection 622
 - TopicConnection 622
 - ConnectionFactory 622
 - TopicConnectionFactory 622
 - Default messaging provider
 - Consuming messages 635
 - Destination 622
 - Queue 622
 - Topic 622
 - Engine 468, 646
 - UUID 649
 - Formats
 - Record-oriented 626
 - Tagged delimited 626
 - XML 626
 - Levels of reliability 648
 - Listener port 633
 - MessageConsumer 622
 - QueueBrowser 622
 - QueueReceiver 622
 - TopicSubscriber 622
 - MessageProducer 622
 - QueueSender 622
 - TopicPublisher 622
 - Performance 630
 - Point-to-point 622, 637, 639, 647
 - Publish/subscribe 44, 622, 647, 655, 672–673, 715, 728, 730–731, 734, 739, 747, 752
 - publish/subscribe 472
 - QueueConnectionFactory 622
 - Session 622
 - QueueSession 622
 - TopicSession 622
 - Simultaneous message processing 718
 - System 625, 650
 - Topology 710, 720
 - WebSphere MQ
 - Consuming messages 632
 - Workload 687
 - Workload patterns 627
 - From MDBs 632
 - From Web/EJB container 627
- Messaging services 466, 476
- Metadata API 921
- Method extensions 993
- Method invocation 853
- Method Statistics 861
- Microsoft Data Access Components 949
- Microsoft IIS *See* Microsoft Internet Information Server
- Microsoft Internet Information Server 13, 1019
- Microsoft Loopback Adapter
 - Add 149
 - Configure 152
- Microsoft Windows 2000 Server 388
- Min connections 984
- MinCommit 1028
- Minimum heap size 1005
- MinSpareThreads 1018
- mod_deflate 1016
- mod_mem_cache 610
- Model View Controller *See* MVC
- Module
 - Performance data 774
- Monitoring
 - Active 832
 - Passive 832
- MPM architecture 1017
- mpm_winnt 1017–1018
- mpm_worker 1017
- MQ
 - JMS connection pooling 722
 - JMS pooling 722
 - Queue manager 722, 726

- Transaction support 712
- MQ Explorer GUI 728
- MQ JMS classes 717, 722
- MQJMS.POOLING.THRESHOLD 722
- MQJMS.POOLING.TIMEOUT 722
- mqsistart 746
- Multicast 471
- Multi-process 1016
- Multi-Processing Modules architecture 1017
- Multithreaded logging library 935
- Multi-tier environment 72
- MVC 510, 541, 615, 896, 901

N

- Name space 347
- NAS 467, 475, 489
- NAT/NAPT 181
- native_stderr.log 1009
- NDAdvisor.java.servlet 193
- Network Appliance 504
- Network Attached Storage *See* NAS
- Network bandwidth 1016
- Network Deployment Manager 12
- Network File System v4 490
- Network interface 141
- Network latency 376
- Network utilization 974
- Never 999
- NFA 103, 118
- NFS v4 490
- nleaf 1028
- nlevels 1028
- Node
 - Performance data 774
- Node Agent 12
- Non-blocking connection 336
- Non-durable subscription 715
- Non-forwarding address 103
- Nonrepeatable reads 994
- Non-serializable data 551
- Nortel Alteon Controller 110
- NotSupported 915, 999–1000
- Number of concurrent endpoints 637
- numRequest 780
- NVRAM 491

O

- Object

- Collection 932
- Creation 931
- Destruction 931
- Loitering 932
- Pool 931
- Reference 930, 932
- Reuse 931
- Serialization 910
- Temporary 932
- Object cache instance 510
- Object pools 779, 969
- Object References Graph 864
 - Object Details view 866
- Object Request Broker 342, 355, 777–778, 977, 987–988, 1022
 - Connection cache maximum 1025
- On demand computing 74
- One-phase commit 666
- Online banking 43
- Online shopping 43, 45
- Online trading 43, 47
- onMessage method 653, 655, 694, 702, 705, 718, 721, 765
- Open System Testing Architecture *See* OpenSTA
- OpenLoad 965
- OpenSTA 897, 948
- Operations ratio 979
- Optimistic concurrency control 384, 922
- Optimistic methods 997
- Optimization level 1027
- Optimizelt 981
- Option A caching 345, 383, 992–993
- Option B caching 345, 383, 992–993
- Option C caching 346, 383, 992–993
- Oracle 676, 689
 - Read committed 994
- Oracle 9i IAS - Relational Cache 505
- ORB 342, 777–778, 977, 987–988, 1022
 - Plug-in 342, 370
 - Failover 371
 - Normal operation 354
 - Service 987
 - Thread pool size 986, 1025
- org.apache.struts.action.ActionServlet.class 538
- OS TCP/IP keep-alive time-out 376
- Out of Memory exception 1005
- Over-utilizing objects 1005, 1009

P

- Package statistics 858
- Page Detailer 525, 839–840, 886
 - Connection Attempt Failed 887
 - Connection Setup Time 887
 - Considerations 890
 - Data capture 887
 - Delivery Time 888
 - Details view 893
 - Detect broken links 890
 - Detect server timeouts 890
 - Host Name Resolution 887
 - Legend 892
 - Measure Web application performance 886
 - Monitoring HTTP and HTTPS requests 886
 - Page Time 887
 - Performance measurement
 - Browser cache 890
 - Network delays 890
 - Packet loss 890
 - Server Response Time 888
 - Socks Connection Time 888
 - SSL Connection Setup Time 888
- Paging 484
- Paging activity 974
- partitionID 281, 297
- Pass by reference 915, 927, 1022
- Pass by value 370, 915
- Passivation 346
- Peak
 - Concurrent users 940
 - Load 969
 - Request rate 940
 - Usage hours 940
- PercentMaxed 987–988
- PercentUsed 668, 983
- PerfMBean 773
- Performance 6, 100, 502
 - Access intent read 995
 - Application design 941
 - Average concurrent users 940
 - Average request rate 940
 - Breaking point 941
 - Caching of data 937
 - Caching of EJB references 914
 - Connection pool 935
 - Data counters 779
 - EJB home 914
 - Garbage collection 1002
 - Hardware capacity 976
 - Hits per day 940
 - Hits per hour 940
 - Hits per minute 940
 - Hits per month 940
 - Inefficient settings 813
 - Load 812
 - Load testing 942
 - Load testing tools 941, 945
 - ApacheBench 945
 - Advantages 946
 - Limitations 945
 - Options 947
 - Mercury LoadRunner 945
 - OpenSTA 948
 - Architecture 950
 - Collectors 954
 - Commander 950
 - Create test 954
 - Define test 953
 - Execute test 955
 - Features 948
 - Name Server 950
 - Prerequisites 949
 - Randomization of requests 953
 - Record a script 951
 - Repository 950
 - Repository host 950
 - Script Modeler 951
 - Specify runtime parameters 954
 - Task group 953
 - View test results 955
 - Rational Performance Tester 945
 - Segue SilkPerformer 945
- Measuring 840
- Minimize memory usage 931
- Monitoring 793
- Monitoring - tuning - testing 971
- Monitoring tools 771
- Overhead 933
- Peak concurrent users 940
- Peak request rate 940
- Peak usage hours 940
- Poor coding practices 941
- Profiling 840
- Project cycle 840
- Recommendations 813
- Reduce EJB overhead 914
- Regular usage hours 940

- Requirements 63
- SDO 923
- Site abandonment rate 940
- Stress testing tools 942
 - Evaluation 942
 - Functions 942
- Strings 936
- Testing 62, 940, 969
 - Data set 63
 - Network 64
 - Phases 64
 - Application testing 64–65
 - Development testing 64
 - Results 63
 - Test environment 63
- Testing tools 964
 - Grinder 965
 - JMeter 964
 - LoadRunner 965
 - OpenLoad 965
 - Segue SilkPerformer 965
 - TestMaker 964
 - TestNetwork 964
 - WebLOAD 965
 - WebStone 965
- Think time 943
- Trade 6 391, 436
- Traffic patterns 941
- Tuning 939
 - Top ten monitoring list 965
- Tuning parameter hotlist 975
- Tuning values 939
- Vertical scaling 933
- Web site performance improvement 890
- Performance Advisor in Tivoli Performance Viewer
 - See TPV Advisor
- Performance Advisors 813, 965, 971
 - Runtime Performance Advisor 813
 - TPV Advisor 813
- Performance analysis 969
 - Load test
 - Measure steady-state 973
 - Ramp-down time 973
 - Ramp-up time 973
 - Production level workload 970
 - Repeatable tests 970
 - Saturation point 970
 - Stress test tool 970
 - Terminology 969
 - Load 969
 - Peak load 969
 - Requests/second 969
 - Response time 969
 - Throughput 969
- Performance bottleneck 841
- Performance data
 - Connection pooling 777
 - Counters 774
 - Enterprise beans 777
 - J2C connectors 777
 - Module 774
 - Node 774
 - Server 774
 - Submodule 774
- Performance data classification
 - Boundary statistic 773
 - Bounded range statistic 774
 - Count statistic 773
 - Range statistic 773
 - Time statistic 774
- Performance Data Framework 772
- Performance data hierarchy 774
- Performance Management 61
- Performance monitoring 790
 - Develop monitoring application 804
 - Under load conditions 770
- Performance monitoring and management tools 803
- Performance Monitoring Infrastructure See PMI
- Performance Monitoring Service 693, 765
- Performance of a Web page
 - Key factors 890
- Performance problems 770
 - Application design 770
 - Application view 770
 - Back-end system 770
 - End-user view 770
 - External view 770
 - Hardware 770
 - Monitoring tools 770
 - Network 770
 - Product bugs 770
 - Response time 770
- Performance Servlet 771
- Performance Trace Data Visualizer 981
- Performance tuning 62
 - Access log 1019
 - AIX 1013

- AIX file descriptors 1013
- AIX ulimit 1013
- AIX with DB2 1013
- DB2 1026
- DB2 Buffpage 1027
- DB2 configuration advisor 1026
- DB2 log files 1013
- DB2 logging 1026
- DB2 MaxAgents 1026
- DB2 MaxAppls 1026
- DB2 MinCommit 1028
- DB2 on Linux 1026
- DB2 query optimization level 1027
- DB2 reorgchk 1028
- Disabling security 1020
- Disk speed 976
- Dynamic cache service 1020
- Full duplex 977
- Java memory 1002
- Life cycle 941
- Logging 1019
- Microsoft IIS 1019
- Network 977
- Number of requests 978
- Ongoing process 941
- Operating System 1012
- ORB 1022
- OS 1012
- Parameter hotlist 975
- Pass by reference 1022
- Process
 - Testing - Evaluating - Tuning 941
- Processing time 978
- Processor speed 977
- Security 1020
- SSL 1021
- SSL hardware accelerator 1022
- Sun ONE Web Server 1019
- System memory 977
- Top-ten monitoring list
 - Average response Time 967
 - CPU utilization 967
 - Datasource connection pool size 967
 - Disk and network I/O 967
 - EJB container thread pool 967
 - Garbage collection statistics 967
 - JVM memory 967
 - Live number of HTTP Sessions 967
 - Number of requests per second 967
 - Paging activity 968
 - Web container tread pool 967
 - Web server threads 967
 - Web server 1015
 - Web server reload interval 1015
 - Windows 1014
 - Windows MaxUserPort 1014
 - Windows TcpTimedWaitDelay 1014
 - XML parser selection 1025
- Persistence
 - Database 78
- Persistent service
 - High availability 491
- Persistent session management 294
 - Database session persistence 411
 - Memory-to-memory session replication 411
- Persistent sessions 26
- Pervasive computing 906
- Pessimistic methods 997
- Phantom reads 994
- PID 849
- Planning for scalability 43
- Planning requirements 40
 - Capacity 40
 - Functional 40
 - Non-functional 40
- Plug-in
 - See ORB
 - Plug-in
 - See Web server plug-in
- Configuration
 - ClusterAddress 268
- Configuration file
 - Custom 78
 - Generation 258
 - Settings 255
- Installation location 262
- Refresh interval 263
- Workload management 17
 - Workload management
 - Plug-in 310
- Plug-in file
 - Propagation 262
- plugin-cfg.xml 81, 252, 258, 432, 569
 - Propagation 262
 - Propagation to Web server 244
- PMI 693, 770–771, 786, 790, 805–806, 813, 965
 - API 771
 - Counters 772

- AllocateCount 695
- Available Message Count 694
- CloseCount 694
- CreateCount 694
- MessageBackoutCount 694
- MessageCount 694
- ServerSessionPoolUsage 694
- Total Messages Consumed 694
- Total Messages Produced 694
- WaitTime 694
- Data 776
 - Garbage collection 786
- Enterprise bean Performance Module 694
- Instrumentation 770, 805
- Instrumentation levels 781
- JCA Connection Pools Performance Module 694
- MBeans 773
- Metrics 773
- Monitoring level 776
- Performance modules 798
- Predefined statistic set
 - All 780
 - Basic 780
 - Custom 780
 - Extended 780
 - None 780
- Sequential counter update 782
- Service 773, 782, 791, 972
 - Enabling using Administrative Console 782
 - Enabling using wsadmin 785
 - List objects 785
- SIB Service Performance Module 693
- Point-to-point messaging 639, 655, 672–673, 705, 730–731, 738, 747, 752
- Pool Size 983
- Port 234
 - Unique port number 235
- Positive weights 269
- Precompilation 985
- Prefer local 358, 366, 405
 - Advantage 366
- prep 1027
- Prepared statement cache 813, 985
 - Size 985
- Presentation layer 898, 906, 916
- PrimaryServers tag 271, 276
- Process affinity 357–358, 360, 365, 368, 370
- Process ID 849
- Process interactions 869
- Processing
 - Up-front 1002
- Processing bottlenecks 978
- Production environment 231
- Production system tuning 66
- Profile 396
 - Custom 396
 - Deployment Manager 396
- Profile creation
 - Custom 400
 - Deployment Manager 397
- Profile Creation Wizard 397
- Profiler 839–840
 - Agent 845
 - Analyzing the execution of an application 868
 - Application process 845
 - Call path 881
 - Class Instance Statistics 859
 - Client 847
 - Code coverage 842
 - Collect Object References 865
 - Configuration settings 855
 - Data collection 854
 - Deployment host 846
 - Development host 846
 - Execution Flow view 867
 - Execution time analysis 842
 - Filters 876
 - Instance Statistics 862
 - Memory analysis 841
 - Memory Leak Analysis 865
 - Method Details view 881
 - Method Invocation view 867
 - Method Statistics 861
 - Object and Class Interaction (Sequence Diagram) 868
 - Object References Graph 864
 - Object Details view 866
 - Output 854
 - Package statistics 858
 - Performance analysis views 880
 - Performance Call Graph view 880
 - Probekit 843
 - BCI engine 843
 - Process Interaction View 871
 - Profiling Monitor view
 - Class interactions 870
 - Host interactions 870

- Object interactions 870
- Process interactions 870
- Thread interactions 870
- Profiling set
 - Execution History 876
- Sequence Diagrams 884
- Test client 845
- Testing methodology 856
- Thread analysis 842
- Thread Interactions View 872
- Using 857
- Views 857
- Profiler agent 786
- Profiling
 - Add application sources 877
 - Filters 852
 - Instances of classes 862
 - Limits 853
 - Method calls 867
 - Methods 861
 - Program execution 868
 - Project 852
 - Set 852
 - Start Monitoring 879
 - Test scenario 879
 - Tools 839–840
- Profiling and Logging perspective 847, 849, 857, 875
- Profiling Monitor view 857
- Profiling tools 898, 933
- Project
 - Life cycle 897
- Project cycle 840
- Proof-of-concept 61
- Propagate plug-in file 262
- Properties file 352
- Provider
 - JDBC 984
- Provider URL 347, 351
- pthread_mutex 614
- Publish/subscribe messaging 44, 472, 647, 655, 672–673, 715, 728, 730–731, 734, 739, 747, 752
- Push frequency 550

Q

- QCF 622, 704
 - QueueConnection 704
- Queue 980, 983

- Connection factory 702
- Destination 654, 702, 704
 - Settings 713
- Local 738
- Manager 652, 699, 701–702
 - Local 734
 - Receiving 738
 - Remote 734
 - Sending 738
- QueueConnection 634, 704, 707
- QueueSession 634, 704
- Queuing before WebSphere 979
- Queuing network 977, 979
- Quorum 476

R

- RAID array 977
- Ramp-down 946
- Ramp-up 946
- Random 267, 271
- Range statistic 773, 780
- Ratio calculation 978
- Rational Application Developer 515, 586, 839–840, 868, 925
 - Application profiling 840
 - Performance measuring 840
 - Performance profiling 840
 - Profiling 841
 - Code coverage 842
 - Data collection engine 844
 - Execution time analysis 842
 - Memory analysis 841
 - Memory Leak analysis 842
 - Probekit 843
 - Thread analysis 842
 - Profiling architecture 844
 - Profiling project 852
- Rational Performance Tester 159, 856, 897, 945, 956–957
 - Create Performance Test 959
- Rational Purify 933
- Rational Robot 832
- Rational TestManager 957
- Read committed 994–995
 - Oracle 994
- read only method 995
- Read timeout 989
- Read uncommitted 994–995

- Read/write ratio 1020
- Read-ahead 999
- readObject() 910
- read-only method 995
- RECOVERY.RETRY.INTERVAL 726
- Redbooks Web site 1047
 - Contact us xxv
- RedHat 614
- RedHat Advanced Server 2.1 1014
- RedHat Enterprise Linux WS 957
- RedHat Linux Advanced Server 2.1 614
- reduce memory usage 993
- Reference
 - Static 933
- reference LookupTime 780
- Reflection 902, 936
- Refresh interval
 - Web server plug-in 263
- regedit 154
- Regular usage hours 940
- Relational database 920
- Reload enabled 991, 1001
- Reload interval 991, 1001
- reorgchk 1028
- Repeatable read 994–995
 - DB2 994
- Repetitive test
 - Memory leak 1006
 - Module level 1006
 - System level 1006
- Replication domain 295, 383, 405, 412, 548
- Request decomposition 832
- Request filtering 1016
- Request Metrics 770–771, 805–806, 808
 - ARM agent 807
 - Enabling using Administrative Console 806
 - Filtering mechanism 805
 - Filters 812
 - EJB method name 806
 - JMS 806
 - Source IP 806
 - URI 806
 - Web Services 806
- Level
 - Debug 807
 - Hops 807
 - None 806
 - Performance_debug 807
- Log 807
- Trace format 808
- Trace record format 809
 - Correlators 809
- Request/response model 909
- Required 915, 999–1000
- Requirements
 - Non-functional 63
- Requires New 915, 999–1000
- Resilience 489
- Resource allocation 1002
- Resource Analyzer 790
- Resource constraint problems 805
- Resource leaks 975
- Resource lock 721
- Resource references 651, 700
- Response filtering 1016
- Response time 6, 43, 777
- RetryInterval 256, 274, 337, 1016
- Reuse
 - Business logic 919
- RFC
 - 2616 505
 - 3143 505
- RMI 986
- RMI over IIOP 773
- RMI/IIOP 908, 914, 987
- RMM 474
- Rollback 671, 728
- Round robin
 - Server weights 267
 - Turn off 267
 - With weighting 267
- Route tag 255
- Routing table 359
 - Windows 149, 153
- runmqsc 747
- runstats 1028
- Runtime delays 1002
- Runtime Performance Advisor 813, 968
 - Advice Configuration 816
 - Advice configuration 819
 - Configuration 815
 - Output 819
 - Using 817

S

- Sample topology 86, 388
- SAN 476, 488–490

- Saturation point 970–971, 980
 - Maximum concurrency point 982
- Scalability 4–5, 21, 74, 100
 - Horizontal and vertical combined 22
 - Horizontal scaling 22
 - Vertical scaling 21
- Scalability planning 43
- Scaling
 - Vertical 933
- Scaling techniques 42, 51
 - Aggregating user data 54
 - Appliance server 53
 - Batch requests 54
 - Caching 55
 - Creating a cluster of machines 52
 - Managing connections 55
 - Segmenting workload 53
 - Using a faster machine 52
- Scaling-out 59
- Scaling-up 59
- Scope 709, 716
 - Application 709
 - Cell 709
 - Cluster 709
 - Node 709
 - Server 709
- SDO 638, 906, 912, 919–920
 - Abstraction layer 920
 - Advantages 923
 - Architecture 920
 - Components
 - Data graph 921
 - Data mediator 921
 - Data object 921
 - Data mediators 920
 - Disadvantages 923
 - Performance 923
- Security 4, 10
 - Cluster member 23
 - SSL communication 11
- Security cache timeout 11, 1021
- Security requirements 45
- Segue SilkPerformer 945, 965
- SELECT 995
- Selector 637, 639
- Separator 281
- Serializable 993–994, 996–997
 - Dirty reads 994
 - Nonrepeatable reads 994
 - Phantom reads 994
- Serialization 370
- Serializing 910, 914
- Server
 - Performance data 774
- Server affinity 30
 - EJB 30
 - Entity bean 30
 - HTTP session 30
 - Stateful session beans 30
- Server certificate 244
- Server cluster 14, 228
 - Workload management 20
- Server failure 466
- Server selection policy 358, 365, 368
- Server tag 255
- Server template 404
- Server weights 18, 267, 269, 359
 - maxWeight 270
- ServerCluster tag 255
- ServerLimit 1018
- Server-sided XSLT processing 906
- Service Data Objects *See* SDO
- Service Integration Bus 643, 645
 - Clustering support 656
- service() method 900, 1020
- Service-oriented architecture 912
- Servlet 2.3 specification 280–282
- Servlet cache instance 510
- Servlet clustering 17
- Servlet response time 792
- Servlet session manager 778
- Servlet workload management 17
- Servlet/JSP result cache 541, 609
- Servlet/JSP result caching 529–530, 585
 - Configuration 531
 - Demonstration 533
- Servlets 883
- ServletToDestination 650–651, 700
- Session
 - Affinity 271, 281, 937
 - Identifier 281
 - Object 900
- Session beans 913
 - EJB
 - Session bean 345
 - Passivation 912
- Session cache 813
- Session clustering 26

- Session configuration
 - Basic 287
 - Level 287
 - Distributed 289
 - General Properties 287
- Session facade 913, 917
- Session ID 281
- Session identifier
 - Clone ID 294, 296
 - cookies 282
 - SSL ID 282, 284
 - URL rewriting 283
- Session management 4, 24–25, 279
 - Behavior 312
 - Create database 291
 - Database persistence 78
 - DRS 26
 - EJB
 - 28
 - Memory-to-memory replication 26, 78, 294
 - Persistent sessions 26, 78, 294
 - Replication domain 295
 - Session clustering 26
 - Writing frequency 300
- Session persistence 78
 - Database 909
 - Database persistence 280
 - Memory-to-memory replication 280
- Session pool
 - Maximum sessions 632
- Session state 10
- Session time-out 285
- SESSIONMANAGEMENTAFFINI 284
- SessionSampleURLRewrite 1034
- setDeliveryMode() 712
- setEntityContext() method 915
- setupCmdLine 677, 749
- Shared resources 933
- SIB service Performance Module 693
- Siebel 922
- Siebel API 922
- Sikatra JProbe 933
- Simultaneous message processing 718
- Single point of failure *See* SPOF
- Single threaded 933
- Single-thread process 1016
- SingleThreadModel 900
- Singleton object 932
- Singleton service 466, 476, 479
- Messaging engine 468
- Transaction Manager 468
- SIT 65
- Site abandonment rate 940
- Site Selector 110
- Sizing 58, 63
- SMIT 132
- smit 1014
- smitty 1014
- SMP 478
- SNMP 955
- SNMP traps 837
- snoop servlet 233, 309
- SOA 912
- SOAP 392, 437, 773
- SOAP/HTTP 908
- SOAP/HTTPS 908
- Socket states 974
- Software
 - Requirements 388
- Source code profiling 830
- Spawned threads 937
- Spawning 937
- Speedera 504
- SPOF 7, 59, 85, 358, 466, 646, 734
 - Database 295
- SQL 985
 - Statement 925, 995
 - Dynamic 1027
 - Static 1027
- SSA 488
- SSL 479, 560
 - Accelerator hardware 11
 - Bulk encryption/decryption 1021
 - Cipher suite 1022
 - Connections 1021
 - Handshake 1021
 - Handshaking 11
 - Hardware accelerator 1022
 - ID 282, 284, 315
 - Performance 1021
 - Session identifier 282
 - Session time-out 285
- Stack traces 937
- StartServers 1018
- Stateful 25
- Stateful session beans 28, 344, 909, 911
 - Failover 344, 377
 - Best practices 383

- Configuration 378
 - Application level 379
 - EJB container level 378
 - EJB module level 381
- Home objects
 - Clustering of 344
- Instance variables 911
- Replication 32
- Server affinity 30
- Size 912
- Stateless 25, 909
- Stateless session beans 28, 344, 914, 917
 - Failover 377
- Statement Cache Size 986
- Static content 502
- Static field 933
- Static reference 933
- Static SQL statements 1027
- Static variable 932
- stats_time 1028
- stdout 934
- Steady memory utilization 1009
- Sticky time 111
- Stored procedures 926, 936
- Stream 266
- String concatenation 936
- Strings 936
 - Manipulation operation 936
- Struts 510, 899, 901, 908, 923
 - Action handlers 908
 - Framework 510
 - JSP UI tag library 901
- struts-config.xml 538
- Subject Matter Expert 831
- Submodule
 - Performance data 774
- Subscription
 - Durable 734
 - Non-durable 674, 731, 734
- Sun Java System Web Server 13
- Sun ONE Web Server 13, 1019
- Supports 915, 999–1000
- Surrogate-Capabilities 568
- SUSE Linux Enterprise Server 957
- SUSE Linux Enterprise Server 8 SP2A 1014
- SVT 65
- Swapping 1011
- Symptom database *See* Log Analyzer
- sync() 300

- Synchronization 933
 - java.util.Hashtable 934
 - java.util.Vector 934
 - Method 934
 - Object 934
 - Overhead 934
- Synchronization points 1006
- Synthetic transaction 805
- Synthetic Transaction Investigator 832
- System Integration Test 65
- System level metrics 778
- System resource management 830
- System test
 - Memory leak 1006
- System Verification Test 65
- System.arraycopy() 937
- System.gc() 1002
- System.out 771
- System.out.println() 934
- SystemOut.log 471, 482, 764, 809, 813
- Systems management framework 830

T

- Target destination 652, 701
- TCF 622
 - Client ID 752
- TCP sockets 1026
- TCP/IP keep-alive 376
- TCP/IP ports 402
- Test applications
 - BeenThere 426
 - Trade 6 438
- Test environment 63–64
- TestMaker 964
- TestNetwork 964
- Think time 943
- Thread pool 55, 778, 792, 983, 987–988
 - Web container 988, 991
- Thread starts 786
- Thread waits 777, 787
- ThreadLimit 1018
- Threads 627, 777–778, 787, 984, 986, 988, 990
 - Idle 1018
 - Maximum number 1018
 - Spawned 937
 - Web server 990
- Threads of work 703
- ThreadsPerChild 1018

- Threshold 931
- Throughput 6, 969
- Throughput curve 980–981
- Tiles 511, 901
 - Framework 511
- Time statistic 774, 780
- TimesTen 505
- Time-to-live 505, 616
- Tivoli 63
- Tivoli Data Warehouse 833
- Tivoli Intelligent Orchestrator 833
- Tivoli Monitoring for Transaction Performance (TMTP) 831
- Tivoli Performance Viewer 668, 693, 718, 724, 765, 770–771, 775, 782, 790–793, 814, 946, 965, 972, 982–983, 985, 987–988, 1005, 1007
 - Change scale of a counter 802
 - Clear Buffer 802
 - Collection Status 793
 - Configuration
 - Log settings 795
 - User settings 794
 - Refresh Rate 795
 - Views 795
 - Configure settings 794
 - Counters 800
 - Resetting 802
 - Engine 791
 - Functionality 793
 - Interface 791
 - Modifying charts 802
 - Monitoring level 798
 - Performance Advisor 794
 - Performance modules 798
 - Record a log file 800
 - Record in log files 800
 - Refresh data 799
 - Replay a log file 801
 - Replay log file 801
 - Summary Reports 794
 - Summary reports 796
 - Connection pool summary 797
 - EJB method summary 797
 - EJB summary 796
 - Servlet summary 796
 - Thread pool summary 797
 - View Performance Modules 794
 - Viewing JVMPI output 789
- Tivoli System Automation 465, 468, 488
- Tivoli Web Health Console 833
- TMTP 831
- Topic destination settings 713
- Topologies
 - Horizontal scaling 82
 - Vertical scaling 82
 - Web server separated 81
- Topology 71
- Topology selection 94
 - Criteria 94
 - Requirements 94
- toString() 932
- Total memory usage 1008
- TPV Advisor 813, 821, 968
 - Output 824
- tr0 141
- tr1 141
- Trace 828
 - Web server plug-in 308
- TraceFormat 828
- Trade 6 436, 644, 672, 698, 730, 840, 961
 - Configuration script 749
 - Home servlet 533
 - Installation 676
 - Using Administrative Console 450
 - Using script 453
 - Installation script 453
 - JMS functionality 673, 731
 - JMS usage example 674, 686
 - PingServletToMDBQueue 702
 - Populate database 458
 - Servlet2MDBQueue 653
 - TradeBrokerMDB 653, 702, 731
 - Uninstall 460
- Trade3 508, 532, 542
 - Benchmark
 - Using command caching 611
 - Using ESI 611
 - Using servlet and JSP result caching 611
 - Without caching 611
 - Performance
 - Using servlet and command caching 614
 - Using servlet, command, and ESI caching 614
 - Without caching 614
- Transaction 28, 370, 383
 - Boundary 927
 - Context 927
 - Flow 805

- Isolation level 995
 - Read committed 994–995
 - Read uncommitted 994–995
 - Repeatable read 994–995
 - Serializable 993, 996
- Rollback 634
- Support 712, 714
- Type 915
 - Mandatory 915
 - NotSupported 915
 - Required 915
 - Requires New 915
 - Supports 915
- Transaction affinity 31, 345, 357–359, 365, 368, 370, 383
- Transaction logs 485, 489, 1028
- Transaction Manager 468, 476, 489, 496, 778
 - Clustered TM Policy 497
 - HA solution 489
 - High availability 485
 - Failover 494
 - Hot-failover 489
 - Using external HA software 496
 - Configuration 496
 - Using shared file system 489
 - Configuration 491
 - WebSphere V5 487
 - Transaction server 42
 - Transaction service 466
 - Failure 486
 - Recovery process 487
 - Transactions per second (tps) ratio 978
 - Transfer Object pattern 915
 - Transport
 - Channels 237
 - Transport buffer 472
 - Transport chains 231
 - Transport protocol 229
 - Transport tag 255
 - Transport type 478
 - Channel Framework 479
 - Multicast 478
 - Unicast 478
 - Transports 234
 - Channels 989
 - Setting up 236
 - Setting up multiple transports 239
 - Settings 235
 - try {} catch{} block 937
 - Tuning 57
 - Tuning failover 335
 - Two-phase commit 666, 714
 - Two-phase commit transaction 485

U

 - UAT 65
 - UDDI 392
 - ulimit 1013
 - UML *See* Unified Modelling Language
 - UML2 Object Interactions sequence diagram 884
 - Unbounded ORB Service Thread Pool Advice 817
 - Unified Modelling Language 868
 - Unit of work 998
 - UNIVERSAL_JDBC_DRIVER_PATH 448, 683, 689, 750, 762
 - Unmanaged node 79
 - unsetEntityContext() 915
 - UPDATE/UPDATE pair 995
 - Up-front processing 1002
 - Uri tag 254
 - UriGroup tag 254
 - URL
 - Encoding 283, 317
 - Rewriting 283, 317
 - Example 319
 - Session identifier
 - URL rewriting 282
 - urctest Web module 1035
 - Usage patterns 66
 - User Acceptance Test 65
 - User interface 912
 - User Interface component *See* UI component
 - User to business 46–47
 - User to data 44
 - User to online buying 45

V

 - valueUnBound() 911
 - Variable
 - Final 932
 - Scope 932
 - Static 932
 - Vector 1007
 - verbosegc 1009
 - Versant enJin 505
 - Vertical scaling 21, 82, 933

- view-helper class 902
- Virtual host 232, 516, 759
 - admin_host 232
 - default_host 232
 - Servlet request 232
 - Wildcard settings 233
- VirtualHost tag 254
- VirtualHostGroup tag 239, 254
- vmstat 1008, 1029

W

- WASService.exe 400
- WCInboundDefault 231, 989
- WCInboundDefaultSecure 231
- Web application
 - Execution 847
- Web application performance 886
- Web application testing 940, 964
- Web client 15
- Web container 55, 228, 347, 389, 651, 700, 988
 - Availability 89
 - Cluster 404
 - Failover 272
 - Failure 272, 321
 - Failure during request 331
 - Failure with active sessions 322
 - HTTP transport 239
 - HTTPS transport 239
 - Requests 979
 - Setting up 230
 - Thread pool 627
 - Threads 627
 - Transport 234
 - Transport chains 231
 - Workload management 388
- Web module
 - Auto-reload 263
- Web performance analysis 969
- Web primitive 764
- Web server 12, 42, 567
 - Access log 339
 - Availability 89, 100
 - Cluster 100, 102
 - Configuration
 - Reload interval 1015
 - Error log 304, 308
 - Failover 83
 - IBM HTTP Server V6 251

- Managed node 79, 246
 - Configuration 247
- Management 244
- Maximum concurrency thread setting 990
- Overloading 100
- Performance 100
- Process-based 1016
- Reload interval 1015
- Scalability 100
- Single point of failure 83
- Thread-based 1016
- Threads 57
- Unmanaged node 79, 248
 - Configuration 249
 - IBM HTTP Server 80
- Workload management 17
- Web Server Node 78
- Web server plug-in 12, 235, 273, 504, 567, 805, 988–989
 - Caching 508
 - Config element 263
 - Configuration file 252
 - Propagation 252
 - Configuration file generation 258
 - Manual 260
 - Configuration service 258
 - Failover 275
 - Failover tuning 335
 - Log file 305
 - Log location 305
 - Logging 304
 - LogLevel 305
 - Stats 306
 - Marking down cluster member 274
 - Normal operation 309
 - Primary and backup servers 275
 - Processing requests 264
 - Refresh interval 263
 - refreshInterval attribute 263
 - Regenerate from Administrative Console 260
 - Regenerate using GenPluginCfg command 261
 - Regeneration 431, 453
 - Retry interval 274
 - RetryInterval 274
 - Suppress load balancing 271
 - Trace 308
 - Troubleshooting 304
 - Workload management 17, 264
 - Workload management policies 267

- Web services 392, 437, 779, 912, 920
 - SOAP 392
 - UDDI 392
 - WSDL 392
- Web services gateway 778
- Web site classification 44, 50
- Web site performance
 - Caching 891
 - Downloads of large objects 891
 - Number of embedded objects 890
 - SSL 891
- Web tier 896
- WebContainer Inbound Chain 13, 73, 231
- WEB-INF 532
- WebLOAD 965
- WebSphere
 - Administrative Console 13
 - Administrative service 13, 773
 - Application server 12
 - Basic configuration 4
 - Cell configuration 402
 - Cluster 19, 244, 395
 - Creation 395
 - Configuration repository 14
 - Deployment Manager 7, 12
 - EJS WLM 23
 - Embedded HTTP transport 13
 - InfoCenter 975
 - JMS 13
 - Load Balancer 91
 - Custom advisors 91
 - Node Agent 12
 - Plug-in 228
 - WLM 17
 - Resource analyzer 335
 - Runtime Messages 813
 - Single Server 342
 - Thread pool 813
 - Topology 71
 - Trade 6 436
 - Variables
 - DB2UNIVERSAL_JDBC_DRIVER_PATH 448, 683, 689, 750, 762
 - Web server plug-in 12
- WebSphere Application Server
 - Application development 896, 930
- WebSphere Application Server Network Deployment V6 388
- WebSphere Application Server topology 720
- WebSphere Application Server V6 342
 - Performance improvements 31
- WebSphere Business Integration Event Broker 697, 732, 734, 736, 740
 - Configuration 742
 - Configuration manager 743
 - Message broker 743
 - Message Brokers Toolkit 743
- WebSphere Command Framework API 541
- WebSphere Commerce Suite 505
- WebSphere configuration
 - Dynamic 9
 - Mixed 9
- WebSphere Connection Manager 55
- WebSphere CORBA CosNaming 347
- WebSphere custom advisor 93
- WebSphere Data Objects 924
- WebSphere Dynamic cache service *See* Dynamic cache service
- WebSphere Edge Components *See* Edge Components
- WebSphere Edge Server 609
- WebSphere Extended Deployment 476, 483
- WebSphere external caching 508
 - Using Caching Proxy 508
 - Using ESI 508
 - Using FRCA 508
- WebSphere High Availability service 374
- WebSphere InfoCenter 975
- WebSphere MQ 628, 697, 699, 701, 713, 722, 732, 834
 - Backout queues 728
 - Backout threshold 728
 - Client-side selection 637
 - Cluster 736–737
 - Clustering 732
 - Workload management 732
 - Components
 - Queue manager 701
 - Configuration 736
 - Connection pooling 630
 - Correlation ID 638
 - Create cluster 740
 - Create queue 741–742
 - Create queue manager 738, 740
 - Dead Letter Queue 738
 - Explorer 738, 740, 764
 - Failover and workload management 734
 - Header 638

- Infrastructure 734
- Message filtering by content 637
- Message ID 638
- Provider 697
- Publish/subscribe broker 743
- Queue
 - Local 738
- Queue manager 637, 699, 702, 712
 - Local 734
 - Receiving 738
 - Sending 738
- Remote queue manager 734
- Server 699
- Server-side selection 638
- Workload management 737
- WebSphere Partition Facility 476, 483
- WebSphere queue size 979–980
- WebSphere Runtime Messages 820
- WebSphere runtime resources 792
- WebSphere sample applications 691
- WebSphere Studio Application Developer 515, 586
- WebSphere Studio Application Monitor *See* WSAM
- WebSphere transaction service 485
- WebStone 965
- Weight
 - Current 270
 - Maximum 269
 - Positive 269
- Weighted round robin 267–268
- Windows Common Internet File System 490
- Windows Service 400
- WLM *See* Workload management
- Workload characteristics 49–50
- Workload distribution policy 255
- Workload management 4, 6, 16, 20, 228, 244, 264, 733, 778, 933
 - Backup servers 228–229
 - BackupServers tag 276
 - Browser requests 268
 - Demonstration 391
 - HTTPS considerations 244
 - Policies 267
 - Random 267
 - Weighted round robin 267–268
- Primary servers 228
- PrimaryServers tag 276
- Random 271
- Selection process 309
- Server cluster 20
- Servlet 17
- Suppress load balancing 271
- Web server 17
- Web server plug-in 17
- Weighted round robin 229
 - Server weights 269
- Workload patterns 44, 674
 - Business to business 48
 - Customer self-service 46
 - Online shopping 45
 - Online trading 47
 - Publish/subscribe 44
 - User to business 46–47
 - User to data 44
 - User to online buying 45
- wsadmin 440, 676, 709, 771, 782
 - Create cache instance 512
 - GenPluginCfg 260–261
 - list PMIService 785
 - modify 786
 - save 786
- WSAM 831, 834
 - Architecture 834
 - Console application 835
 - Data collectors 835
 - Managing server 835
 - Portal Monitoring 837
- WSDL 392, 437
- wsOptimisticRead 996
- wsPessimisticRead 996
- wsPessimisticUpdate 996
- wsPessimisticUpdate-Exclusive 996
- wsPessimisticUpdate-noCollision 996
- wsPessimisticUpdate-WeakestLockAtLoad 996
- WYSIWYG 899

X

- XA support 712, 714
- XA transactions 721
- XML 524, 626, 905
 - Cache policy file 541
 - Data source 920
 - Data Transfer Objects 906
 - Parser 1025
- Xnoclassgc 1012
- XrunpmiJvmpiProfiler 787, 790
- XSL standard 905
- XSL stylesheets 906

XSLT 626, 901
 Processing
 Server-sided 906
 Transformation 905
XSLTC 906

Z

z/OS 835
zLinux 835

Archived

Archived

IBM



Redbooks

WebSphere Application Server V6 Scalability and Performance Handbook

(2.0" spine)
2.0" <-> 2.498"
1052 <-> 1314 pages



Redbooks

WebSphere Application Server V6 Scalability and Performance Handbook

WebSphere Handbook Series

Workload manage Web server, servlet, and EJB requests

Learn about performance monitoring and tuning

This IBM Redbook discusses various options for scaling applications based on IBM WebSphere Application Server Network Deployment V6. It explores how a basic WebSphere configuration can be extended to provide more computing power by better exploiting the power of each machine and by using multiple machines. It examines a number of techniques:

- Using the IBM WebSphere Edge Components Load Balancer to distribute load among multiple Web servers
- Using the WebSphere Web server plug-in to distribute the load from one Web server to multiple application servers in a server cluster
- Using the WebSphere EJB workload management facility to distribute load at the EJB level
- Using dynamic caching techniques to improve the performance of a Web site
- Using the HAManager to meet high availability needs of critical applications
- Using application development best practices to develop a scalable application
- Using the performance tuning options available with WebSphere to adjust the application server configuration to the needs of your application

This redbook provides step-by-step instructions for implementing a sample, multiple-machine environment. We use this environment to illustrate most of the IBM WebSphere Application Server Network Deployment V6 workload management and scalability features.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks