WebSphere® software

IBM

WebSphere Application Server Network Deployment V6: High Availability Solutions

- WebSphere Handbook Series
- Explore WebSphere HA options
- Learn about external clustering solutions

Birgit Roehm
Adeline Chun
William Joly
Tim Klubertanz
Li-Fang Lee
Hong Min
Yoshiki Nakajima
Nagaraj Nunna
Terry O'Brien
Kristi Peterson
Jens Rathgeber
Michael Schmitt

Redbooks

ibm.com/redbooks

International Technical Support Organization

WebSphere Application Server Network Deployment V6: High Availability Solutions

October 2005

SG24-6688-00

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

First Edition (October 2005)

This edition applies to IBM WebSphere Application Server Network Deployment V6.0.1.

Contents

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®	Domino®	OS/400®
@server®	DB2 Universal Database™	Parallel Sysplex®
Redbooks (logo) ™	DB2®	POWER™
developerWorks®	Enterprise Storage Server®	POWER5™
iSeries™	FlashCopy®	Rational®
i5/OS™	HACMP™	Redbooks™
pSeries®	Informix®	SupportPac™
z/OS®	IBM®	Sysplex Timer®
zSeries®	IMS™	Tivoli®
AFS®	MQSeries®	TotalStorage®
AIX 5L™	MVS™	WebSphere®
AIX®	NetServer™	
CICS®	OS/390®	

The following terms are trademarks of other companies:

Enterprise JavaBeans, EJB, Java, JavaBeans, JDBC, JMX, JSP, JVM, J2EE, Solaris, Sun, Sun Microsystems, SunPlex, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries 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.

VERITAS is a trademark or registered trademark of VERITAS Software Corporation or its affiliates in the U.S. and other countries

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

Preface

This IBM® Redbook discusses the high availability aspects of IBM WebSphere Application Server Network Deployment V6 and high availability of related components, such as the Web servers or directory servers.

This book discusses in detail:

- High availability concepts.
- WebSphere Application Server clustering considerations, the failover process, the WebSphere® HAManager, and WebSphere component's reactions to failures.
- High availability system administration, such as application management, hardware replacement or upgrade, and software upgrades.
- WebSphere Node Agent and Deployment Manager high availability using external clustering software solutions such as IBM HACMP™, IBM Tivoli® System Automation (TSA), VERITAS Cluster Server, and Sun™ Cluster.
- High availability considerations and differences when using WebSphere on iSeries™ and zSeries®.
- End-to-end WebSphere system high availability involving WebSphere MQ, Web servers, Load Balancer, firewalls, and LDAP servers.

The book also gives an introduction into how to backup and recover a Network Deployment configuration.

For information about how high availability is provided for the WebSphere V6 default messaging provider, see Chapter 12, "Using and optimizing the default messaging provider" of *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392.

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 (ITSO), Raleigh Center.

Figure 1 From left to right: Jens Rathgeber, Adeline Chun, Nagaraj Nunna, Yoshiki Nakajima, Michael Schmitt, Li-Fang Lee, Terry O'Brien, Kristi Peterson, Birgit Roehm, Tim Klubertanz, and William Joly. Not pictured: Hong Min

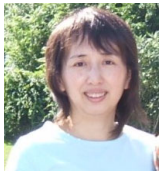
Birgit Roehm is a Project Leader at the ITSO, Raleigh Center. She writes redbooks and teaches workshops about various aspects of WebSphere and Domino®. Before joining the ITSO in 2003, Birgit worked in iSeries Advanced Technical Support, Germany, and was responsible for Domino and WebSphere on iSeries.

Adeline Chun is a certified IT Specialist working for IBM WebSphere Technical Sales in Canada. Prior to joining the technical sales team, she worked as an IT architect for IBM Lab Services for WebSphere, implementing WebSphere e-Business solutions for worldwide banks, including HSBC, Banco Santigo, ING, ICBC Shanghai, Barclays Bank UK, ANZ Bank, and BBL Belgium. Using the WebSphere architecture, she implemented Teller, Call Center, and Internet banking solutions. Currently, she helps the sales teams to deliver technical implementation of J2EE™ applications based on WebSphere across industries, with expertise in WebSphere Application Server, Edge Components, and Enterprise Modernization solutions.

William Joly is an IT Architect consultant at LivInfo, a French IT Services Society based in Noisy-le-grand, close to Paris. He has nine years experience in Java™ software development. William is architecting WebSphere Application Server projects for customers and helping them in all project phases. He is certified in DB2® and WebSphere products.

Tim Klubertanz has six years of experience with IBM and is currently working in the Systems and Technology Group IBM @server® Custom Technology Center. He holds a bachelors degree in Computer Science from Luther College and has experience with WebSphere application design and implementation on multiple platforms, including iSeries and pSeries®. His current role focuses on iSeries availability solutions for customers, including clustering, storage, and WebSphere Application Server.

Li-Fang Lee is a test strategist working for the WebSphere Test and Quality Organization in Rochester, MN. Her current focus area is high availability. She works to understand customer business requirements, to design customer-like test scenarios across the organization, and to lead a team to carry out the test scenario to ensure high availability of WebSphere Application Server.



Hong Min (pictured at left) is an IT specialist at the IBM Design Center for business on demand, Poughkeepsie, USA. She has eight years of experience helping customers enabling e-business using IBM technologies. Her technical interests include WebSphere, J2EE applications, Grid computing and zSeries, and so forth.

Yoshiki Nakajima is an IT Specialist at IBM Systems Engineering Co. Ltd (ISE), part of the ATS function in Japan. He has five years of experience with IBM and IBM subsidiary, having worked within both Global Services and Technical Support. He has experience in the Java and WebSphere fields and provides technical support in WebSphere. Lately, he is interested in Web Services and Enterprise Service Bus.

Nagaraj Nunna is a Managing Consultant and IT Architect at IBM Global Services, USA. He has been working with IBM since 2001. His areas of expertise include distributed computing and middleware, in general. Specifically, he has worked with Enterprise Application Integration (EAI), XML, and Java 2 Enterprise Edition (J2EE), performance and tuning, and integration with WebSphere Application Server.

Terry O'Brien is a software engineer from Rochester, MN. He has 22 years of experience in software engineering with IBM. Terry holds a degree in Computer Science from Michigan Technological University and a Master in Business Administration from Winona State University. His areas of expertise include WebSphere Application Server, high availability software, XML programming, business-to-business, and Domino for iSeries.

Kristi Peterson is a software engineer from Rochester, MN. She has four years of experience in the field of software testing. Kristi holds a degree in Computer Science and English from Luther College in Decorah, Iowa. Her areas of expertise include WebSphere Application Server, software testing, high availability software, test application development, documentation review, and scenario testing development.

Jens Rathgeber is a Senior Architect at SerCon GmbH (IBM subsidiary) in Mainz, Germany. He has more than seven years of experience in the IT field and has been involved in several customer engagements involving WebSphere for the last five years. His areas of expertise include performance tests, operations architecture, and project management.

Michael Schmitt is a Staff Software Engineer in Rochester MN and has worked for IBM since 2000. He holds a Bachelor of Science degree from Winona State University in Computer Science. He has experience in Java and WebSphere Application Server, primarily developing code for the Client Container, Platform Messaging, and the Workload Manager components. His current role at IBM is working for the WebSphere Application Server High Availability Center of Competency, whose primary goal is to provide high availability best practice information using WebSphere Application Server.

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

Carla Sadtler, Margaret Ticknor, Jeanne Tucker
ITSO, Raleigh Center

Alex Louwe-Kooijmans, Patrick Ryan
ITSO, Poughkeepsie Center

Thomas Gray, Joanna Pohl-Miszczyk
ITSO, Rochester Center

Douglas Berg, Mark Bransford, Siva Guntaka, James Habinek, Billy Newport, Wayne Rosario, Darin Scherer, Jim Stopyro
IBM Rochester

Lori Adington, Rohith Ashok, Jakob Mickley, Rengan Sundararaman
IBM Raleigh

Michael Schwinck, Mainz, and Markus Mueller, Boeblingen
IBM Germany

Jeff Anders
Sun Microsystems™, Inc.

Lowell Shulman
VERITAS Software Corporation

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 will team with IBM technical professionals, Business Partners, 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.

Obtain 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 e-mail 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

Part 1

High availability concepts

1

Understanding high availability concepts

Maintaining high levels of access to information across heterogeneous environments, without compromising a quality user experience, can challenge any IT organization.

This chapter introduces some of the WebSphere high availability concepts, the different levels of WebSphere end-to-end high availability, and WebSphere high availability solutions planning.

1.1 Process availability and data availability

There are two kinds of high availability: process high availability and data high availability.

In a IBM WebSphere Application Server Network Deployment V6 environment, you can have one or more of each of the following processes:

- WebSphere Deployment Manager process
- WebSphere Node Agent process(es)
- WebSphere Application Server process(es)
- HTTP server process(es)
- Load Balancer process(es)
- Database server processes
- Firewall processes
- LDAP server process(es)
- WebSphere MQ process(es)
- Networked file system (for example NFS, AFS®) process
- Operation system processes

Even if you make all of these processes highly available, the WebSphere system might still fail due to data availability. Without data availability, the WebSphere system cannot do any meaningful work. Therefore, we include a short discussion on how to make the WebSphere data management systems highly available in this book.

Data management is an integral part of a WebSphere production system and is needed for storing data such as:

- Application data for the application servers.
- The administrative repository for the WebSphere administrative servers (in the form of XML files).
- Persistent session data for WebSphere HTTP sessions if not using memory-to-memory session replication.
- Persistent data for Entity EJBs.
- The default messaging provider data store.
- WebSphere security data for the LDAP servers.
- Transaction log files and other log files.
- WebSphere system and application binaries.
- HTML and image files.

1.1.1 Clustering for high availability

Clustering is a fundamental approach for accomplishing high availability. IBM WebSphere Application Server Network Deployment V6 offers a built-in application server clustering function and the HAManager for protecting WebSphere singleton services. Clustering application servers provides workload management (WLM) and failover for applications that reside on the application server cluster. For more information about application server clustering and failover, see Chapter 2, “WebSphere Application Server failover and recovery” on page 35 and *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392. For details about the HAManager, see Chapter 6, “WebSphere HAManager” on page 175.

In addition, other clustering techniques can be used for WebSphere end-to-end system high availability. The WebSphere system can include a database, an LDAP directory server, firewalls, a Caching Proxy, one or more Load Balancers, HTTP servers, and application servers. For more information about high availability options for components such as the firewalls and Load Balancer, see Chapter 15, “WebSphere end-to-end high availability” on page 561.

Data access is a very important part of most applications, especially transactional applications. In order to achieve 99.x% of WebSphere system availability, we need to integrate platform-specific or application-specific clustering solutions with WebSphere to meet the high availability needs for critical applications. However, a failover of a clustered database server might take minutes to finish if IP-based cluster failover is used.

When using a clustered IBM WebSphere Application Server Network Deployment V6 environment, WebSphere process failures usually do not contribute much to the total number of client request failures, as the WebSphere process failover is instantaneous. Be aware, however, that a clustered database server failover (IP-based cluster failover) might take minutes. To minimize the potential downtime, you can use parallel database servers that are provided by features such as Database Partitioning for DB2 UDB Enterprise Server Edition V8 or Oracle Real Application Clusters (RAC).

There are two kinds of cluster failovers:

- IP-based cluster failover, such as IBM High Availability Cluster Multi-Processing for AIX® 5L™ (HACMP), IBM Tivoli System Automation (TSA), Sun Cluster, and VERITAS Cluster Server.
- Non-IP cluster failover, such as WebSphere WLM and WebSphere HAManager.

Usually, IP-based cluster failover is slower (one to five minutes), and non-IP cluster failover is very fast (instantaneous). Whereas the WebSphere V6

HAManager does not require extra software, most non-IP cluster failover still relies on cluster software such as HACMP, TSA, Sun Cluster, or VERITAS Cluster Software to provide the cluster information. For more information about this topic, see Part 5, “Using external clustering software” on page 283. The concept of IP-based cluster failover and configuration of WebSphere for a clustered environment is explained in Chapter 9, “Configuring WebSphere Application Server for external clustering software” on page 285. In addition, this part of the book covers the following cluster software:

- Chapter 11, “WebSphere and IBM HACMP” on page 417
- Chapter 10, “WebSphere and IBM Tivoli System Automation” on page 367
- Chapter 12, “WebSphere and VERITAS Cluster Server” on page 445
- Chapter 13, “WebSphere and Sun Cluster” on page 483

1.2 Availability definition

Before we describe different high availability (HA) implementations for WebSphere systems, we first need to define high availability and discuss how to measure high availability. Availability is a measure of the time that a server is functioning normally, as well as a measure of the time the recovery process requires after the system fails. In other words, it is the downtime that defines system availability. This downtime includes both planned and unplanned downtime.

Let A be an index of system availability expressed as a percentage, MTBF the mean time between failures, and MTTR the maximum time to recover the system from failures. Thus, we have:

A = MTBF/(MTBF + MTTR)

As MTBF gets larger, A increases and MTTR has less impact on A. As MTTR approaches zero, A increases toward 100%. This means that if we can recover from failures very quickly, we have a highly available system. The time to recover a system includes fault detection time and system recovery time. Therefore, clustering software uses fault detection mechanisms and automatically fails over the services to a healthy host to minimize the fault detection time and the service recovery time. MTTR is minimized because the fault detection time is minimized and no repair attempt is needed. Therefore, A is significantly raised. Any repairs to the failed node and any upgrades of software and hardware will not impact the service availability. This is the so-called *hot replacement* or rolling upgrade.

The availability issue is not as simple as the formula discussed above. First, MTBF is just a trend. For example, if a CPU has an MTBF of 500,000 hours, it does not mean that this CPU will fail after 57 years of use. In reality, this CPU can fail at any time. Second, there are many components in a system, and every

component has a different MTBF and MTTR. These variations make system availability unpredictable using the formula above. We can build a simulation model for an end-to-end WebSphere system's availability with a random process theory such as Markov chains, but this topic is beyond the scope of this book.

For a WebSphere production system, the availability becomes much more complicated, because a WebSphere production system includes many components, such as firewalls, Load Balancers, Web servers, application servers and administrative servers (Node Agent and Deployment Manager), the administrative repository, log files, the persistent session database, application database or databases, and LDAP directory server and database. System availability is determined by the weakest point in the WebSphere production environment.

Usually, redundant hardware and clustering software are used to achieve high availability. Our goal is to minimize the MTTR through various HA techniques. That is, if MTTR=0, then A=100%, no matter what the MTBF is. Using this approach, system availability becomes predictable and manageable.

1.2.1 Levels of availability

First of all, availability is closely related to cost, as shown in Figure 1-1 on page 8. It is important to balance the downtime with cost. Normally, the more you invest, the less downtime there is. Therefore, it is also very important for you to evaluate what you will lose if your WebSphere service is temporarily unavailable. Different businesses have different costs of downtime, and some businesses, such as financial services, might lose millions of dollars for each hour of downtime during business hours. Costs for the downtime include not only direct money.

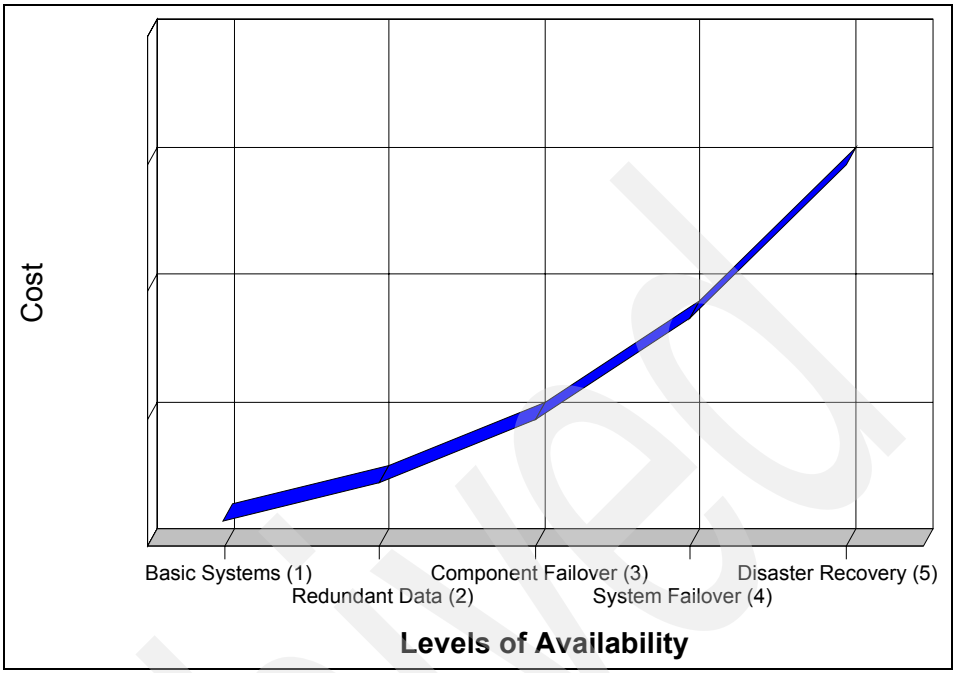


Figure 1-1 Levels of availability and costs

Redundant hardware and clustering software are approaches to high availability. We can divide availability at the following levels:

1. *Basic systems*. Basic systems do not employ any special measures to protect data and services, although backups are taken regularly. When an outage occurs, the support personnel restores the system from the backup.
2. *Redundant data*. Disk redundancy or disk mirroring are used to protect the data against the loss of a disk. Full disk mirroring provides more data protection than RAID-5.
3. *Component failover*. For an infrastructure such as WebSphere, there are many components. As discussed earlier, an outage in any component can result in service interruption. Multiple threads or multiple instances can be employed for availability purposes. For example, if you do not make the firewall component highly available, it might cause the whole system to go down — worse than that, it might expose your system to hackers — even though the servers are highly available.

   IBM WebSphere Application Server Network Deployment V6 provides process high availability (using vertically scaled application server clusters) and process and node high availability (using horizontally scaled clusters). Highly available data management is critical for a highly available

transactional system. Therefore, it is very important to balance the availability of all components in the WebSphere production system. Do not overspend on any particular component, and do not underspend on other components either. For example, for the system shown in Figure 1-2, the system availability seen by the client would be 85%.

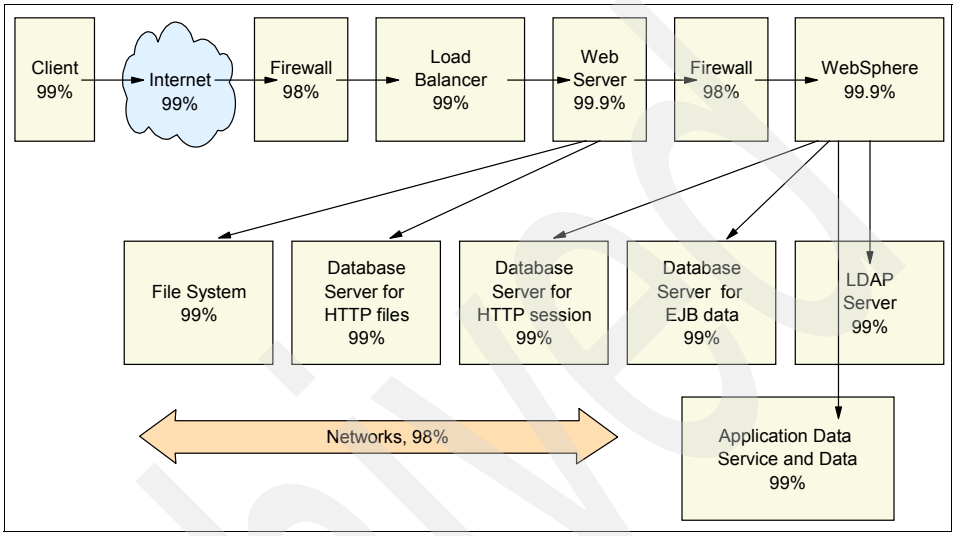


Figure 1-2 Availability chains

4. *System failover.* A standby or backup system is used to take over for the primary system if the primary system fails. In principle, any kind of service can become highly available by employing system failover techniques. However, this will not work if the software is hard-coded to physical host-dependent variables.

   In system failover, clustering software monitors the health of the network, hardware, and software process, detects and communicates any fault, and automatically fails over the service and associated resources to a healthy host. Therefore, you can continue the service before you repair the failed system.

   You can configure the systems as Active/Active mutual takeover or Active/Passive takeover. Although the Active/Active mutual takeover configuration increases the usage of hardware, it also increases the possibility of interruption, and hence reduces the availability. In addition, it is not efficient to include all components into a single cluster system. You can have a firewall cluster, an LDAP cluster, WebSphere server cluster, and database cluster.

Tip: You can also use system failover for planned software and hardware maintenance and upgrades.

5. *Disaster recovery*. This applies to maintaining systems in different sites. When the primary site becomes unavailable due to disasters, the backup site can become operational within a reasonable time. This can be done manually through regular data backups, or automatically by geographical clustering software.

Continuous availability means that high availability *and* continuous operations are required to eliminate all planned downtime.

1.2.2 Availability matrix

We all talk about uptime, and everybody wants 100% uptime. In reality, a 100% uptime system is prohibitively expensive to implement. For some applications, 99% uptime is adequate, leaving a downtime of 14 minutes per day on average (see Table 1-1). For some applications, 99.9% or higher uptime is required. Many people refer to 99%, 99.9%, 99.99%, and 99.999% as *two nines*, *three nines*, *four nines*, and *five nines*. The *five nines* is generally thought of as the best achievable system with reasonable costs, and many vendors offer such solutions. Examples for these solutions are:

- IBM with WebSphere WLM and clustering, WebSphere MQ Cluster, HACMP on AIX, TSA, or the Database Partitioning feature in DB2 UDB Enterprise Server Edition
- Sun Microsystems with Sun Cluster on Solaris™
- VERITAS with VERITAS Cluster Server

Table 1-1 Availability matrix - nine rule

9s	Percentage of uptime	Downtime per year	Downtime per week	Downtime per day
	90%	36.5 days	16.9 hours	2.4 hours
	95%	18.3 days	8.4 hours	1.2 hours
	98%	7.3 days	3.4 hours	28.8 minutes
Two 9s	99%	3.7 days	1.7 hours	14.4 minutes
	99.5%	1.8 days	50.4 minutes	7.2 minutes
	99.8%	17.5 hours	20.2 minutes	2.9 minutes
Three 9s	99.9%	8.8 hours	10.1 minutes	1.4 minutes

9s	Percentage of uptime	Downtime per year	Downtime per week	Downtime per day
Four 9s	99.99%	52.5 minutes	1 minute	8.6 seconds
Five 9s	99.999%	5.3 minutes	6 seconds	864 milliseconds
Six 9s	99.9999%	31.5 seconds	604.8 milliseconds	86.4 milliseconds
Seven 9s	99.99999%	3.2 seconds	60.5 milliseconds	8.6 milliseconds
Eight 9s	99.999999%	315.4 milliseconds	6 milliseconds	0.9 milliseconds

The five nines availability allows a downtime of 864 milliseconds per day, 6 seconds per week, and 5.3 minutes per year as shown in Table 1-1 on page 10. For all clustering techniques with IP takeover, a typical database failover takes two to three minutes. Thus, MTTR equals 2.5 minutes. We, therefore, need an MTBF of 183 days to achieve 99.999% availability. That means only two failovers per year.

Some businesses require 7x24x365 availability, while others require 6x20 or 5x12 availability. The latter do not reduce the requirement for high availability if the business requires the minimum interruption during its business hours. Because we do not know when outages will happen, clustering techniques can keep MTTR short and increase available time even if a business operates only 5x12.

Even though clustering techniques can keep a service highly available, service performance might degrade after the failure occurs until the failed system rejoins the cluster after repair.

Therefore, we suggest describing availability using three factors:

- System uptime percentage
- Business operation hours and pattern
- Performance availability requirement

You should design a high availability system to satisfy the uptime requirement during operation hours and to meet the performance availability requirement.

Most business applications do not require 7x24, so software and hardware upgrades can be performed in the scheduled maintenance time. For the business that requires 7x24 services, clustering techniques provide rolling upgrades and hot replacements by failing over manually from one system to another. See Chapter 4, “High availability system administration” on page 121 and Chapter 5, “High availability application administration” on page 141 for more information.

1.2.3 Causes of downtime

The causes of downtime can be either planned events or unplanned events. Planned events can account for as much as 30% of downtime. As mentioned before, rolling upgrades and hot replacements can reduce the planned downtime. However, the most important issue is how to minimize the *unplanned downtime*, because nobody knows when the unplanned downtime occurs and all businesses require the system to be up during business hours.

Studies have shown that software failures and human error are responsible for a very high percentage of unplanned downtime. Software failures include network software failure, server software failure, and client software failure. Human errors could be related to missing skills but also to the fact that system management is not easy-to-use.

Hardware failures and environmental problems also account for unplanned downtime, although by far not as much as the other factors. Using functions such as state-of-the-art LPAR capabilities with self-optimizing resource adjustments, Capacity on Demand (to avoid overloading of systems), and redundant hardware in the systems (to avoid single points of failure), hardware failures can be further reduced. You can find more information about LPAR for the IBM @server iSeries and pSeries systems in the following resources:

- *Logical Partitions on the IBM PowerPC: A Guide to Working with LPAR on POWER5 for IBM @server i5 Servers*, SG24-8000
- *Advanced POWER Virtualization on IBM @server p5 Servers: Introduction and Basic Configuration*, SG24-7940

You can find information about Capacity on Demand at:

`http://www.ibm.com/servers/eserver/about/cod/`

Many environmental problems are data center related. Having a locally located standby might not suffice, because the entire site environment might be affected. Geographic clustering and data replication can minimize downtime caused by such environmental problems.

The end-to-end WebSphere high availability system that eliminates a single point of failure for all parts of the system can minimize both planned and unplanned downtime. We describe the implementation of such a WebSphere high availability system throughout this book.

1.2.4 Possible single points of failure in the WebSphere system

Table 1-2 lists potential single points of failure in the WebSphere system and possible solutions.

Table 1-2 Possible single points of failure in the WebSphere system

Failure point	Possible solutions
Client access	Multiple ISPs.
Firewalls	Firewall clustering, firewall sprayer, HA firewall.
Caching Proxy	Backup Caching Proxy system.
HTTP sprayer (such as WebSphere Edge Components' Load Balancer)	HA solution of vendor, for example backup Load Balancer server.
Web server	Multiple Web servers with network sprayer, hardware-based clustering.
WebSphere master repository data, log files	HA shared file system, Network File System (NFS), hardware based clustering.
WebSphere Application Server	WebSphere Application Server Network Deployment - application server clustering: ► Horizontal ► Vertical ► Combination of both Additionally for EJBs: backup cluster.
WebSphere Node Agent	Multiple Node Agents in the cluster, OS service, hardware-based clustering. **Note:** The Node Agent is not considered a single point of failure in WebSphere V6. The Node Agent must be running when starting the application server on that node so the application server can register with the Location Service Daemon (LSD). In WebSphere V6 the LSD is HAManager enabled therefore you only need one running Node Agent in the cluster to provide the LSD when the application servers are started on the node. The Node Agent must also be running when changing security related configuration or you might not be able to synchronize with the Deployment Manager later on any more. Refer to Chapter 3, "WebSphere administrative process failures" on page 103 for more information.

Failure point	Possible solutions
WebSphere Deployment Manager	OS service, hardware-based clustering, backup WebSphere cell. **Note:** The Deployment Manager is not considered a single point of failure in WebSphere V6. You need it to configure your WebSphere environment, to monitor performance using the Tivoli Performance Viewer, or to use backup cluster support. Unless these functions are needed, you can run a production environment without an active Deployment Manager. Refer to Chapter 3, "WebSphere administrative process failures" on page 103 for more information.
Entity EJBs, application DB	HA DBs, parallel DBs. **Note:** Make sure your application catches StaleConnectionException and retries, see 15.4, "Database server" on page 574 for more information.
Default messaging provider	WebSphere application server clustering: HAManager provides failover.
Default messaging provider data store	Clustering, data replication, parallel database.
Application database	Clustering, data replication, parallel database.
Session database	Memory-to-memory replication, DB clustering, parallel database.
Transaction logs	WebSphere application server clustering: HAManager provides failover, shared file system with horizontal clustering.
WebSphere MQ	WebSphere MQ cluster, combination of WebSphere MQ cluster and clustering.
LDAP	Master-replica, sprayer, HA LDAP (clustering).
Internal network	Dual internal networks.
Hubs	Multiple interconnected network paths.
Disk failures, disk bus failure, disk controller failure	Disk mirroring, RAID-5, multiple buses, multiple disk controllers.
Network service failures (DNS, ARP, DHCP, and so forth)	Multiple network services.
OS or other software crashes	Clustering, switch automatically to a healthy node.

Failure point	Possible solutions
Host dies	WebSphere application server clustering, hardware-based clustering: automatically switch to a healthy node.
Power outages	UPS, dual-power systems.
Room/floor disaster (fire, flood, and so forth)	Systems in different rooms/different floors.
Building disasters (fire, flood, tornado, and so forth)	Systems in different buildings.
City disasters (earthquake, flood, and so forth)	Remote mirror, replication, geographical clustering.
Region disasters	Put two data centers far away with geographical clustering or remote mirroring.
Human error	Train people, simplify system management, use clustering and redundant hardware/software.
Software and hardware upgrades	Rolling upgrades with clustering or WLM for 7x24x365, planned maintenance for others.

1.2.5 HA technologies for WebSphere system components

As you have seen in Table 1-2 on page 13, there are many different options available to make your WebSphere system highly available. Figure 1-3 is a graphical representation of these possible options for the client tier, the DMZ, and the application tier (hardware failures or disasters as mentioned in the table are not covered in the figures of this section).

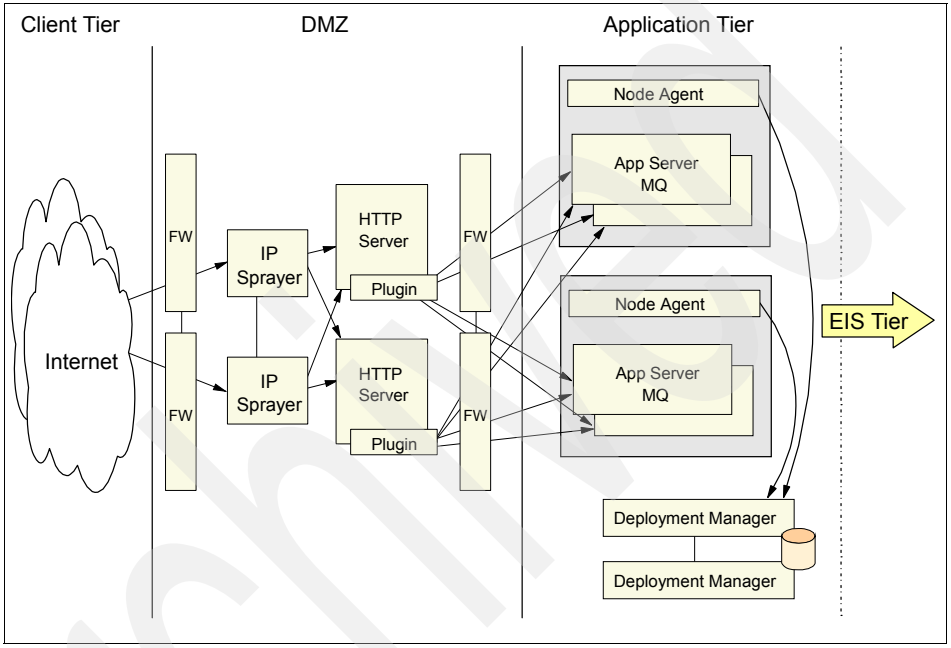


Figure 1-3 HA technologies Client Tier - DMZ - Application Tier

Note: WebSphere MQ can be located in either the application tier or the EIS tier or in both.

Figure 1-4 provides the same information for the EIS tier. The graphic shows an overlap with the application tier for easier comprehension.

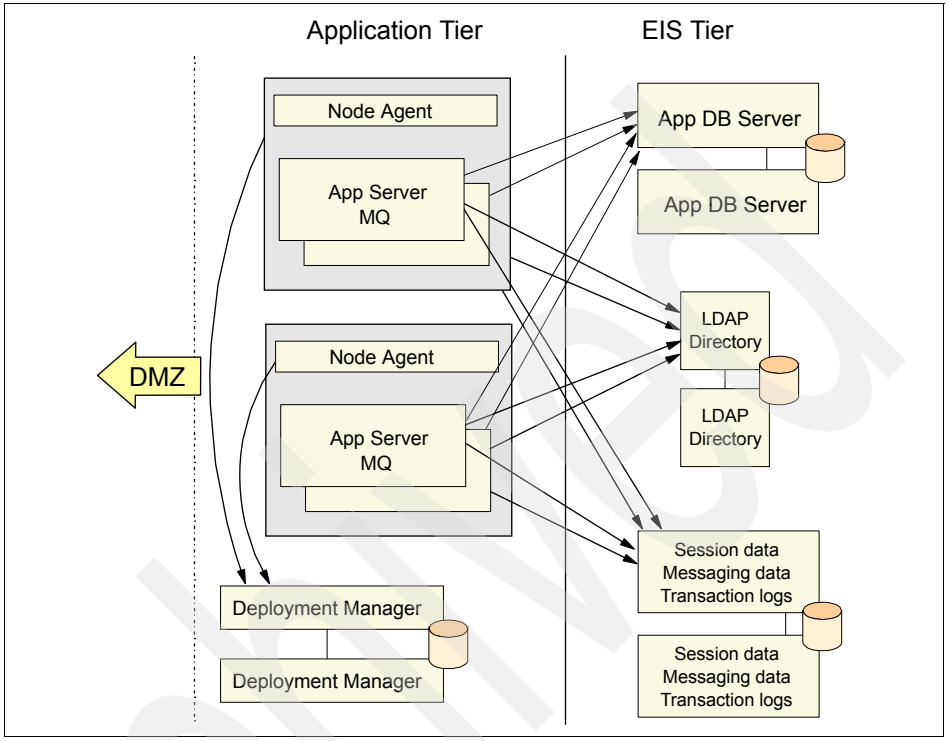


Figure 1-4 HA technologies Application Tier - EIS Tier

1.2.6 Levels of WebSphere system availability

We can deploy WebSphere systems with different redundant hardware, software, networks, processes and components. For the convenience of our discussion, we can roughly divide the deployment into several availability levels. Features discussed for a level are always available for all higher levels also.

Notes - HA levels 1 to 4:

- A DMZ is not shown in the diagrams for WebSphere HA levels 1 to 4. If you want to establish a DMZ, we recommend to move the HTTP server onto a separate system (or LPAR) rather than collocating it with the application server.
- For simplicity reasons, we did not add an LDAP directory server in the level 1 to 4 diagrams. You can either collocate your LDAP directory on (one of) the system or add a separate system.
- We refer simply to the *database* for these four HA levels. The database can, however, include application data, session data, ME data stores, and so on. Therefore, a failure in the database can affect different areas of your WebSphere Application Server environment. Refer to the "Possible single points of failure" section of each HA level for the list of data that can become a single point of failure for this configuration.

The transaction logs reside on the file system. When using horizontal clustering (HA levels 3 and higher), this should be a shared file system which supports lease based locking to allow other cluster members to recover in-flight transactions of a failed application server. For more information, see "WebSphere system HA level 3" on page 20 and 6.7, "Transaction Manager high availability" on page 201.

WebSphere system HA level 1

For WebSphere system HA level 1, as shown in Figure 1-5 on page 19, the HTTP server, WebSphere Application Server, and the database are installed in a single system. There is one application server available to serve the requests. If the host machine dies or crashes, all WebSphere services will be unavailable. A failure of any component will also cause the WebSphere service to become unavailable.

This level can be used by developers and for sites where downtime is not critical.

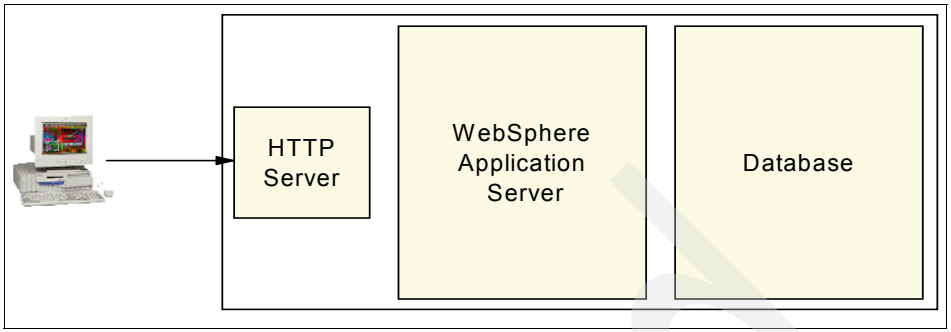


Figure 1-5 WebSphere system HA level 1

Possible single points of failure include the following:

- HTTP server
- Application server
- Database (application data, ME data store)
- Firewalls, LDAP (not shown in Figure 1-5)
- Hardware, which includes any hardware component in the system, such as network cards, network cables, power supply, and so forth, if they are not redundant

WebSphere system HA level 2

You can create an application server cluster (with two or more cluster members) to overcome application server process failures. This is WebSphere system HA level 2, as shown in Figure 1-6 on page 20.

In the case of an application server failure, the WebSphere HAManager makes sure the transaction logs of the failed server are recovered by another application server in the cluster. As all application servers are on the same system and thus can share the available disks, there is no need for additional hardware, such as a SAN.

If you use the default messaging provider in your application, the HAManager also ensures that the messaging engine (ME) is restarted in another cluster member.

Important: You need to make sure that your system is powerful enough to host the additional application server as well as the additional administrative servers (Deployment Manager and Node Agent). Refer to “Vertical scaling” on page 40 for additional information about vertical scaling.

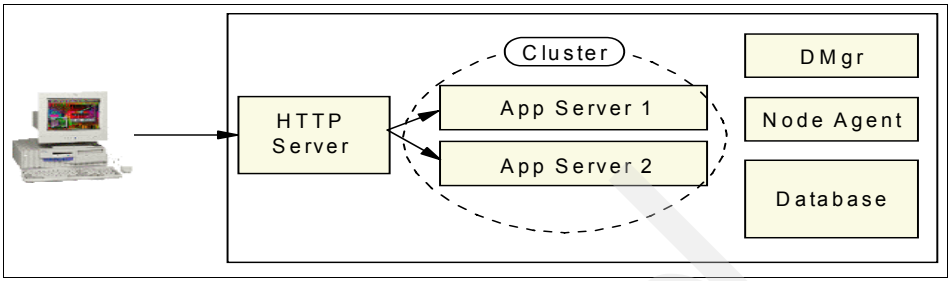


Figure 1-6 WebSphere system HA level 2

Possible single points of failure include the following:

- HTTP server
- Administrative servers (especially Node Agent)
- Database (application data and session data)
- Firewalls, LDAP (not shown in diagram)
- Hardware

Note: Using memory-to-memory session replication instead of a persistent session database eliminates the session data as a single point of failure. Refer to Chapter 1 and Chapter 6 of the *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392 for more information about requirements (such as memory) and configuration, as well as recommendations on when to use which option.

WebSphere system HA level 3

For WebSphere HA level 3, a Load Balancer, multiple HTTP servers, WebSphere vertical and horizontal scaling, a database server hosting the database, plus a shared file system that supports lease based locking for the transaction logs are used, as shown in Figure 1-7 on page 21.

The transaction logs can be recovered by the application servers in System2 should System1 fail. The system hosting the database (application data, and so forth) and the shared file system is, however, still a possible single point of failure.

Note: Transaction logs are only an issue for 2PC transactions. You do not need a shared file system if you do not have 2PC transactions.

As was the case for the previous level, the HAManager takes care of restarting the messaging engine (ME) for the default messaging provider on another application server in the cluster should the application server hosting the ME fail.

With IBM WebSphere Application Server Network Deployment V6 you have the option to run mixed version cells which allows you to upgrade the WebSphere code (apply fixes or install WebSphere new versions) on the various application server nodes in your cell one after the other without interrupting service.

In addition, you might be able to take advantage of the new application rollout update feature which allows you to update an application without bringing down the entire application server cluster. Instead, this function updates one node after the other. That is, it brings down all application servers on the first node, updates the application, restarts the application servers, then continues with the next node. Therefore, the rollout update can only be used with horizontal scaling, not with vertical scaling as used in HA level 2. This function cannot be used for all types of application updates and is not suited for all environments, therefore, for more information refer to 5.4.2, “Rollout update (new feature of WebSphere V6)” on page 153.

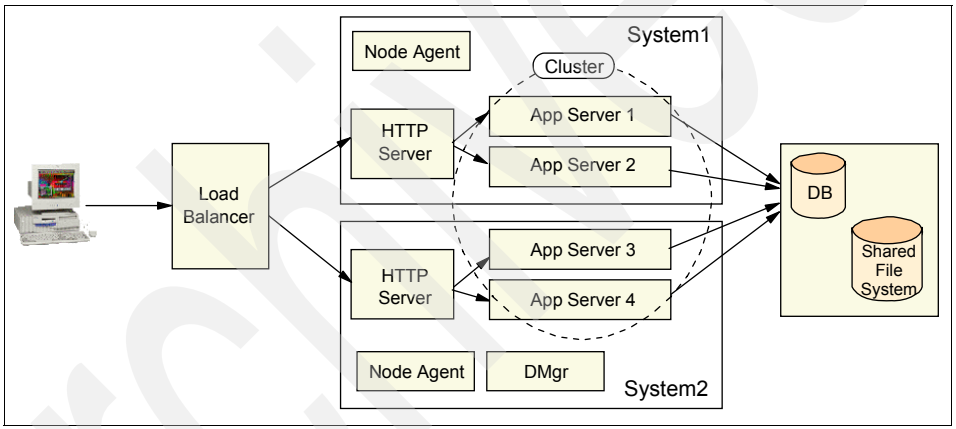


Figure 1-7 WebSphere system HA level 3

Possible single point of failure include the following:

- HTTP sprayer (for example WebSphere Edge Components’ Load Balancer)
- Administrative servers
- Database system (application data, session data)
- Firewalls, LDAP (not shown in diagram)
- Hardware

Important: As the HTTP sprayer/Load Balancer is the entry point into your WebSphere system, we strongly recommend to make it highly available. A cost effective solution is to collocate the backup Load Balancer with one of your HTTP servers. Refer to Chapter 5, “Using IBM WebSphere Edge Components,” in *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392 for detailed information about how to configure a backup Load Balancer.

Figure 1-7 on page 21 shows a scenario using a shared file system that supports lease based locking for transaction log recovery. If your file system does not support this, then you need to do one of the following:

- Restart the failed application server, using either an automatic or manual restart or using clustering software. See Part 5, “Using external clustering software” on page 283 for more information about using clustering software.
- Perform manual peer recovery.
- Disable locking in the transaction service and prevent overloading and network partitioning.

For more information, see *Transactional high availability and deployment considerations in WebSphere Application Server V6* at:

http://www.ibm.com/developerworks/websphere/techjournal/0504_beaven/0504_beaven.html

WebSphere system HA level 4

For WebSphere system HA level 4, as shown in Figure 1-8 on page 24, single point of failure is eliminated for most components of the WebSphere system in this example:

- There is a backup Load Balancer available - as mentioned earlier, the backup can be collocated with one of the HTTP servers.
- There are two (or more) HTTP servers available.
- The combination of horizontal and vertical scaling for the application servers provides failover capabilities for both process failures and hardware failures.
- The database (session data, ME data stores, application data, and so forth) can failover to a second database system using a clustering software such as IBM HACMP, IBM TSA, Sun Cluster, VERITAS Cluster Server, and others.
- The transaction logs reside on the same system as the database. However they are stored in a shared file system that supports lease based locking which is replicated to the backup database system and can also failover to the second system.

The administrative servers (Node Agents and Deployment Manager) remain the only possible single points of failure:

- A failure of the Deployment Manager means that you are not able to use the Tivoli Performance Viewer (which is included in the Administrative Console in WebSphere Application Server V6) and to perform configuration changes - which are rarely done in production environments anyway. Also, the Deployment Manager is needed for a failover when using the backup cluster functionality (see 2.7, “Backup cluster support” on page 87 for more information). Thus a temporarily failure of the Deployment Manager is normally not a big problem for a production environment (some customers even run their production environment without the Deployment Manager being active).

  A simple option is to make sure that the Deployment Manager is restarted automatically after a process failure or a restart of the system is to add it as an OS service, as explained in Chapter 3, “WebSphere administrative process failures” on page 103.

- A failure of the Node Agent has a slightly higher impact as discussed in 3.3, “Node Agent failures” on page 111.

Also, see Part 5, “Using external clustering software” on page 283 for information about using WebSphere with external clustering software. The chapters in this part explain how you can make the Deployment Manager, Node Agents, and application servers highly available using various flavours of clustering software.

Important: We do not show the DMZ and LDAP directory servers in the diagrams up to this HA level. You need to remember that the firewalls and LDAP server must be made highly available as well if you want to avoid any single points of failure in your environment. Refer to Chapter 15, “WebSphere end-to-end high availability” on page 561for more information about HA options for these components.

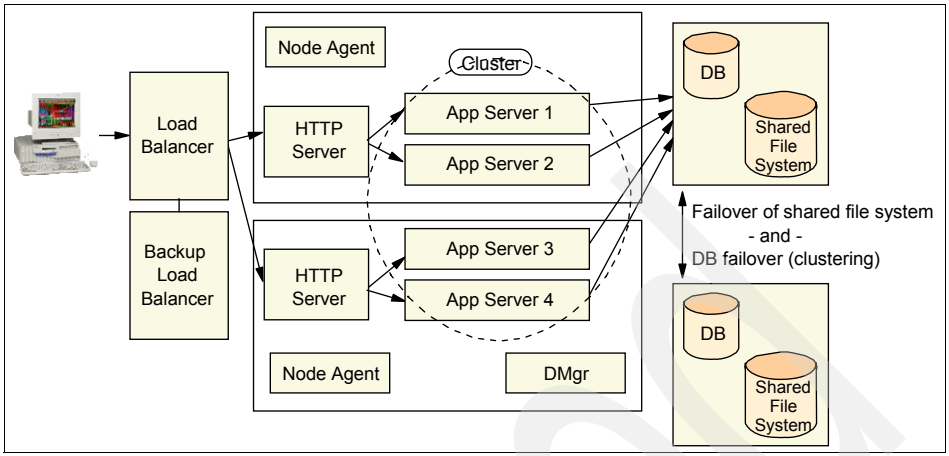


Figure 1-8 WebSphere system HA level 4 - Database clustering/failover

This HA level does not protect you from a site disaster such as a flood, fire, or an earthquake. The only way to have that level of protection is to go to HA level 5 which includes a backup data center.

WebSphere system HA level 5

For WebSphere system HA level 5, two data centers are used for disaster recovery. As shown in Figure 1-9 on page 26, one data center is in the primary service site, and the other data center is a backup. When one site goes down, the backup site can serve requests after an interruption of under one hour to up to a few hours, depending on the configuration and setup of the backup site.

Important: Workload management and failover is not supported between different cells. Therefore, the switch between the primary and the backup site always requires user intervention.

The backup data center hosts a second cell with an identical configuration to the primary cell. The backup cell is *not* serving any requests. The backup cell is configured using the same cell name, same node, cluster and application server names, same resource names, with the same applications installed, and so on. Only the IP addresses of the systems are different. When the primary data center is out-of-service, the DNS is changed to reflect the different IP addresses of the backup environment.

It might take just minutes to get your backup data center up and running. The time it actually takes depends on your setup. For example, if your systems are not up but need to be powered on first, it will take longer. Another option is to have the systems up but the software is not started (cold start).

The option that gives you the shortest failover time is a warm start, which means that all your systems and applications are up and running. The application servers are started using the No Operation policy (NoOP) for the transaction logs and MEs as explained in Chapter 9, “Configuring WebSphere Application Server for external clustering software” on page 285. In this case, in addition to changing the DNS, you need to do the following:

1. Mount the file systems.
2. Issue a script to change the NoOP policy to the normal policies for the transaction logs and MEs.

For this HA level, Figure 1-9 on page 26 illustrates the full environment, including the DMZ, LDAP directory server, WebSphere MQ, and so on.

Important: One box in the diagram does not necessarily mean one system (one piece of hardware). Many components can be collocated on the same system or within different LPARs of one physical system.

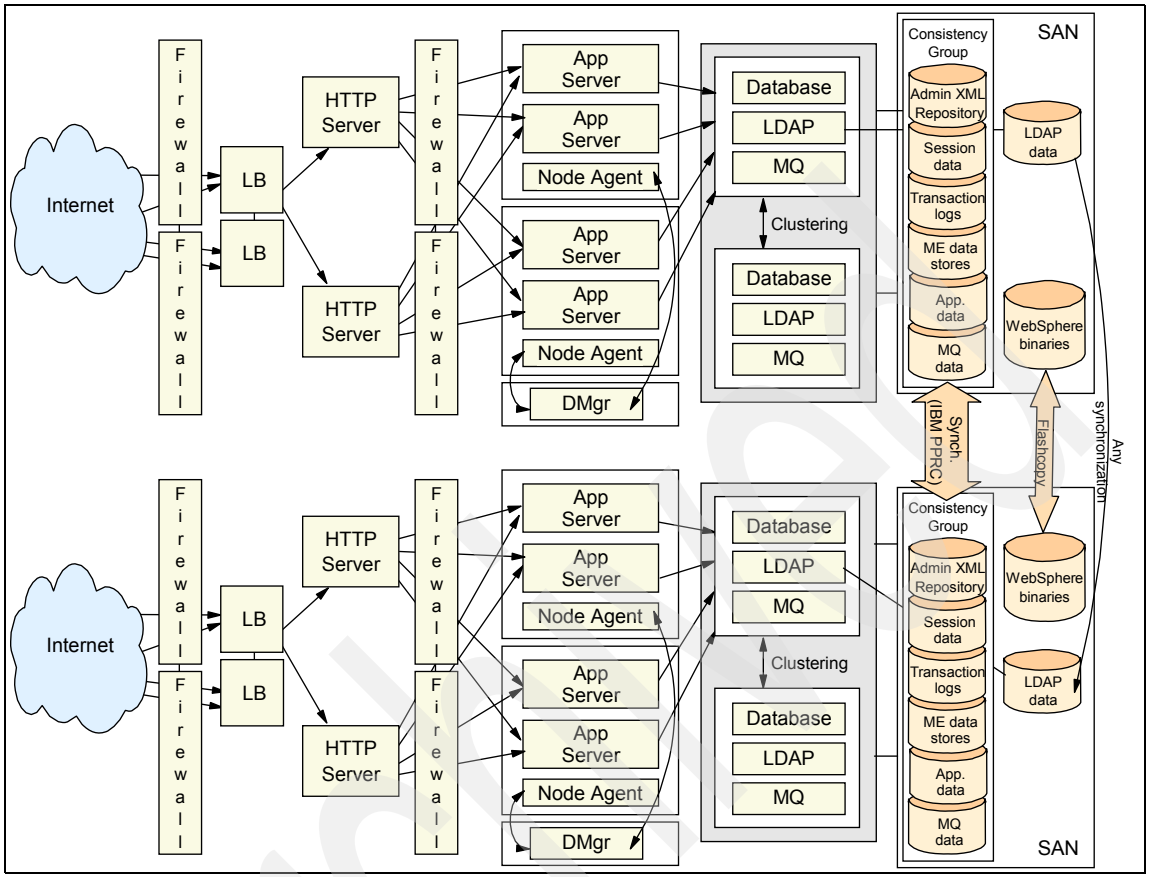


Figure 1-9 WebSphere system HA level 5

The scenario exploits the following features:

- All firewalls have a backup.
- The Load Balancer is configured with a backup.
- There are two (or more) HTTP servers.
- The combination of horizontal and vertical scaling for the application servers provides failover capabilities for both process failures and hardware failures.
- Using clustering software such as IBM HACMP or IBM TSA, the database can failover to a second database system. WebSphere MQ and the LDAP directory server are also protected under the clustering software.
- The data (WebSphere administration repository, session data, transaction logs, ME data stores, application data, MQ data, and so forth) is located on a SAN system. All data is part of one consistency group which is synchronized

to the backup SAN system using, for example, Peer-to-Peer Remote Copy (PPRC). This ensures that all data is in sync should a failover be necessary.

- The WebSphere binaries are installed on the SAN too but are not part of the consistency group. This is not needed because the binaries are changed rarely and do not need to be in sync with the production data such as transaction logs and application data.

  Tip: You can use the FlashCopy function to copy the binaries to the backup SAN system. Using this approach allows you, for example, to apply fixes or to update WebSphere to a new version. Should you find problems you can go back to the old level. When the new version/level is successfully tested, you issue the FlashCopy and thus make sure that the backup cell is up-to-date without the need of applying the fixes or installing the new version in the backup cell.

  See *IBM TotalStorage Enterprise Storage Server Implementing ESS Copy Services in Open Environments*, SG24-5757 for details on PPRC and FlashCopy®.

- The LDAP data is also located on the SAN and can be replicated to the backup SAN system. As with the WebSphere binaries, the LDAP data must not be in sync with the WebSphere environment data (and changes less often) and can therefore reside outside of the consistency group.
- You could also add the Trace logs to the shared disk to ensure problem determination after a site failure.

1.2.7 Planning and evaluating your WebSphere HA solutions

We recommend that you use the following steps when planning and evaluating your end-to-end WebSphere HA solutions:

1. Analyze your requirements:
   - Do you need continuous operations? Most customers do not need continuous operations; therefore, upgrades of software and hardware can be performed offline.
   - What kind of availability do you need during your hours of operation?
   - What is the performance requirement during a failover? For example, the performance might be impacted because fewer nodes or servers serve client requests after a node/server failure. If this is not desired, you need to provide ample extra resources.
2. Analyze the cost factors. How much will you lose when your system is unavailable, and how much can you invest in your system availability? If

downtime is costly, you should invest an appropriate amount to improve your system availability.

3. Estimate the setup and administrative complexity. More complicated systems require skilled people and more configuration and administration effort.
4. Consider all the components in an end-to-end WebSphere system. Usually, the overall availability is dominated by the weakest point of the end-to-end WebSphere system chain. Consider the possibility of a failure in each component and its failover time.
5. Analyze failover time, which mainly includes fault-detection time and recovery time. For different failover mechanisms/techniques, the failover time is different.
6. Analyze the recovery point, where your processing resumes after a failover. It is directly related to the amount of work that is lost during a failover.
7. Understand the programming models. This concerns whether the failover is transparent to clients and whether one component's failover is transparent to other components in an end-to-end WebSphere system. Some services are able to perform failover transparently, because the failover is masked from the application. Others have options for adding application code that can retry when necessary.
8. Know that there is usually more than one solution to address a given failure. Some solutions might have special restrictions. Analyze the trade-offs between the different solutions.

1.3 Failover terms and mechanisms

As mentioned before, an object or an application includes two distinct aspects: functions and data. Therefore, we have process availability and data availability. If a function is not associated with individual data or states, it is easy to achieve high availability by simply restarting this function process when the old process crashes. However, the reality is that functions are associated with individual data or state, some with persisted data in database or files, such as Entity EJBs. We need to make data management systems highly available to all processes and ensure data integrity because the failed process might damage data.

People use the term *failover* for different failovers and different failover mechanisms. We distinguish the following failover mechanisms:

- Failover
- Fail back, fallback
- Fail fast
- Fail transparent

Failover

Failover refers to a single process that moves from the primary system to a backup system in the cluster. This failover process takes several minutes after the fault is detected. This approach can be used for both function-centric or data-centric applications for both Active/Passive and Active/Active configurations. You can also use mutual failover which means that both hosts are running a (different) application and can failover to the other system.

Fail back, fallback

Fail back or fallback is similar to failover, but occurs from the backup system to the primary system when the primary system is back online. For mutual failover, because the backup node has also its original application running, failing back will improve the performance of both applications.

Note: You can find more information about failover and fail back or fallback in Chapter 9, “Configuring WebSphere Application Server for external clustering software” on page 285.

Fail fast

Fail fast refers to uncoordinated process pairs: the backup process is pre-existent, as shown in Figure 1-10. This approach is suitable only for service-centric applications; it cannot be used for data-centric applications.

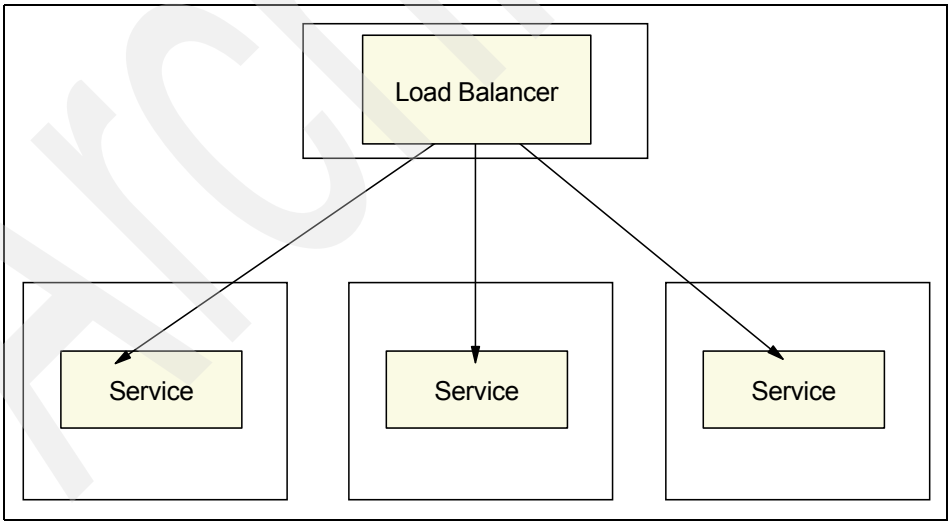


Figure 1-10 Fail fast without clustering

Note the following points:

- These processes belong to the *same* application.
- These processes are not coordinated; in other words, these processes do not know each other's running state.
- These processes are pre-existent on hosts, which is different from the failover case where a new process starts with the takeover of resources after the original process fails.
- These processes do not take resources from each other.

As mentioned, fail fast cannot be used in data-centric applications. It relies on 1-n mapping (such as sprayers) to handle client traffic and process errors. Tuning the connection and TCP/IP timeout parameters is a key part of this kind of failover performance. You need to balance normal running performance and failover performance. Too short a connection and TCP/IP timeout might improve the failover performance, but can harm the normal running performance.

Fail fast is a good solution for your HTTP servers, Caching Proxies, directory servers, and so forth, as it does not only provide high availability but also ensures performance and scalability. See Chapter 15, "WebSphere end-to-end high availability" on page 561 for more information.

Fail transparent

Fail transparent is defined as coordinated process pairs: we have a primary process and a backup process running on separate processors, as shown in Figure 1-11 on page 31. The primary process sends checkpoints to the backup process. If the primary process fails, the backup process takes over. This mechanism is based on the backup process anticipating the failure of the primary process and then taking over without affecting the work in progress or user sessions. This approach requires that each process knows the states of other processes as well as the state of the environment. Therefore, a cluster management layer is required and the application must query the status from this cluster layer. In other words, the processes are coordinated, and errors are handled transparently.

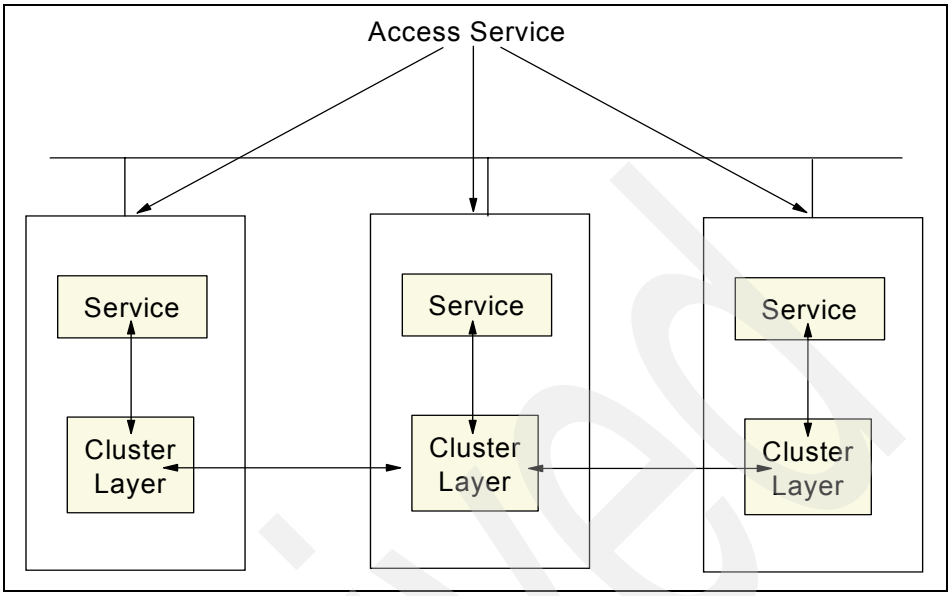


Figure 1-11 Fail transparent with clustering

Summary

In summary, we have three approaches to achieve high availability:

- Process and dependent resource group takeover

  This approach requires cluster information and can be used for both data-centric and service-centric applications. Its implementation is difficult.

- Multiple uncoordinated processes

  This approach does not require cluster information. However, this approach cannot be used for data-centric applications because it does not support the resource group concept.

- Multiple coordinated processes

  This approach requires cluster information, and its implementation is very difficult. It can be used for both data-centric and service-centric applications. It can achieve transparent failover without interruption.

Part 2

WebSphere clustering for HA and HA administration

2

WebSphere Application Server failover and recovery

WebSphere Application Server failover and recovery is realized through the workload management (WLM) mechanism that are provided by IBM WebSphere Application Server Network Deployment V6. If you do not use WLM with your WebSphere Application Servers (and you do not use any other clustering software), your system cannot provide failover support. In this case, your Web or Java clients will fail if your WebSphere Application Server fails.

This chapter gives an introduction to the WLM and failover capabilities of IBM WebSphere Application Server Network Deployment V6. For more details on this topic, refer to *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392.

Topics discussed in this chapter include:

- Introduction to availability
- WebSphere Application Server clustering
- WebSphere workload management defined
- Managing session state among servers
- Web container clustering and failover
- EJB container clustering and failover
- Backup cluster support

2.1 Introduction to 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. Whereas vertical scalability (multiple application servers on one system) 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 or LPARs.

2.1.1 Hardware-based high availability

Using a WebSphere Application Server multiple machine or LPAR configuration eliminates a given application server process as a single point of failure. In IBM WebSphere Application Server Network Deployment V6, there are basically no dependencies on the administrative server process for security, naming, and transactions. Thus, a single process failure normally does not disrupt application processing.

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 of the Deployment Manager only impacts the ability to change the cell configuration, to run the Tivoli Performance Viewer (which is included in the Administrative Console in WebSphere V6), or to perform a failover for EJBs using a backup cluster (see 2.7, "Backup cluster support" on page 87).

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. Some customers even choose to run their production environment without an active Deployment Manager. See Part 5, "Using external clustering software" on page 283 for more information about using clustering software to make the Deployment Manager highly available.

2.1.2 Workload management

IBM WebSphere Application Server Network Deployment V6 workload management optimizes the distribution of incoming requests between application servers that are able to handle a client request. WebSphere workload management is based on application server clusters containing multiple application servers, so-called cluster members. An application deployed to a cluster runs on all cluster members concurrently. The workload is distributed based on weights that are assigned to each cluster member. Thus more powerful machines receive more requests than smaller systems.

Workload management (WLM) also takes care of failing over existing client requests to other, still available application servers and of directing new requests only to available processes should an application server in the cluster fail. In addition, WLM enables servers to be transparently maintained and upgraded while applications remain available for users. You can add additional cluster members to a cluster at any point, providing scalability and performance if an existing environment is not able to handle the workload any more. For more details, see 2.3, “WebSphere workload management defined” on page 43.

2.1.3 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 proportionally 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 might not be completely transparent. For example, a client that is currently in the middle of an operation when a server fails might receive an error from that operation, and might be required to retry (at which point the client would be connected to another, still available server). Or the client might 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 becomes available again, the system might 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% 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* often refers to a physical machine. However, WebSphere also allows for one server process on a given machine to fail independently, while other processes on that same machine continue to operate normally.

2.1.4 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 such as transaction services or 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. See Chapter 6, “WebSphere HAManager” on page 175 for details.

2.1.5 Session management

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).

You can find details in 2.4, “Managing session state among servers” on page 45.

2.2 WebSphere Application Server clustering

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 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.

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

Figure 2-1 on page 40 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.

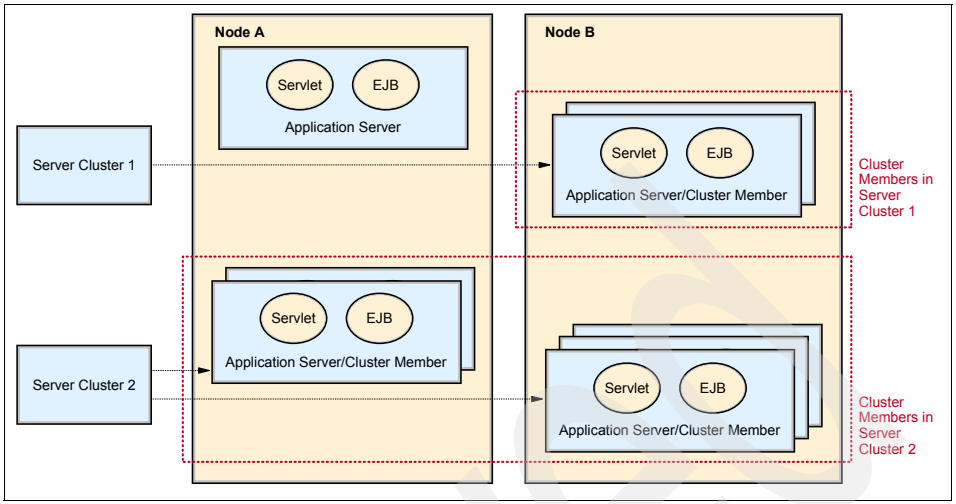


Figure 2-1 Server clusters and cluster members

2.2.1 Clustering for scalability and failover

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

Vertical scaling

In *vertical scaling*, shown in Figure 2-2 on page 41, multiple cluster members for an application server are defined on the same physical machine, or node, which might 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 might still want to have more than one cluster member on the machine for other reasons, such as using vertical clustering for process availability. If a JVM reaches a table/memory limit (or if there is some similar problem), then the presence of another process provides for failover.

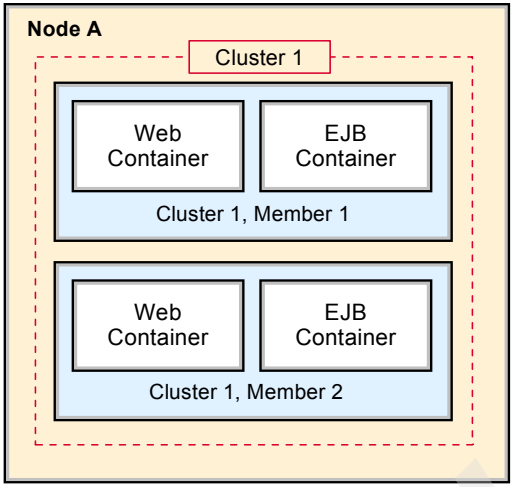


Figure 2-2 Vertical scaling

We recommend that you avoid using *rules of thumb* when determining the number of cluster members for a given machine. The only way to determine what is correct for your environment and application 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.

Horizontal scaling

In *horizontal scaling*, shown in Figure 2-3 on page 42, cluster members are created on multiple physical machines (or LPARs). 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 important benefit of horizontal scaling. If a machine becomes unavailable, its workload can be routed to other machines containing cluster members.

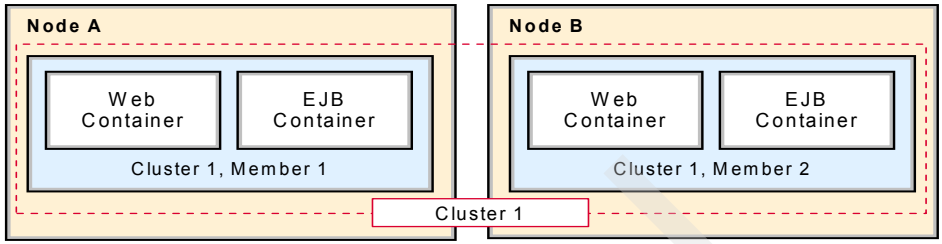


Figure 2-3 Horizontal scaling

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.

Combining vertical and horizontal scaling

WebSphere applications can combine horizontal and vertical scaling to reap the benefits of both scaling techniques, as shown in Figure 2-4.

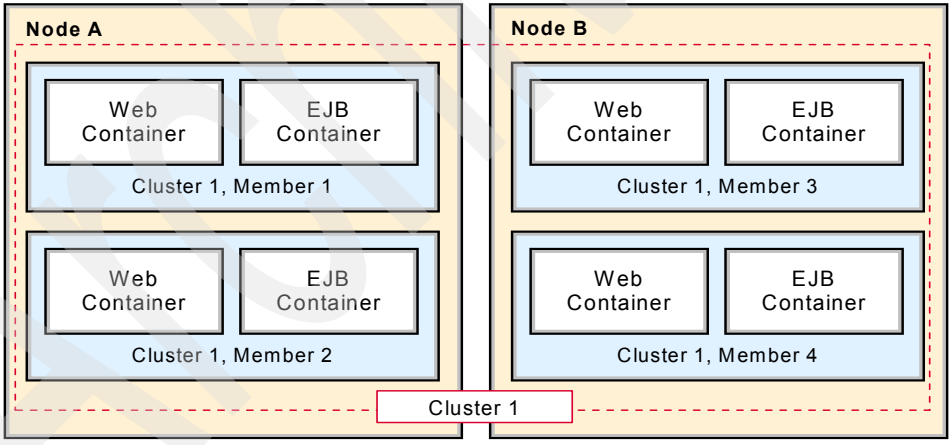


Figure 2-4 Vertical and horizontal scaling

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.

2.3 WebSphere workload management defined

Workload management is implemented in IBM WebSphere Application Server Network Deployment V6 by using application server clusters and cluster members. These cluster members can all reside on a single node (system) or can be distributed across multiple nodes (or LPARs).

You might have 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.

Workload management (WLM) is the WebSphere facility to provide load balancing and affinity between application servers in a WebSphere clustered environment. It optimizes the distribution of processing tasks in the WebSphere Application Server environment; incoming work requests are distributed to the application servers 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. 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.

WLM is most effective when the deployment topology is comprised of application servers on multiple machines, because 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.

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. When an HTTP request reaches the HTTP server, a decision must be made. Some requests for static content might be handled by the HTTP server. Requests for dynamic content or some static content will be passed to a Web container running in an application server. Whether the request should be handled or

passed to WebSphere is decided by the WebSphere Web server plug-in, which runs in-process with the HTTP server. We refer to this as *Plug-in WLM*. For these WebSphere requests, high availability for the Web container becomes an important piece of the failover solution. See 2.5, “Web container clustering and failover” on page 51 for more information.

- *EJB requests* can be distributed across multiple EJB containers. When an EJB client makes calls from the Web container or client container or from outside, the request is handled by the EJB container in one of the clustered application servers. If that server fails, the client request is redirected to another available server. We refer to this as *EJS WLM*. See 2.6, “EJB container clustering and failover” on page 65 for additional information.

2.3.1 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.

Thus 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.

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.

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.

This section only gives you an introduction into WebSphere WLM. The available WLM policies and how requests are distributed among available servers is described in great detail in Chapter 6 and Chapter 7 of *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392.

2.3.2 Benefits

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 2.2, “WebSphere Application Server clustering” on page 39 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.

2.4 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 whether two requests are processed on the same server. 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. However, 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 might 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.

2.4.1 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 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 Web server 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 Web container to retrieve the current session object.

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.

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.

Database persistence

Storing session information in a database, sometimes referred to as *persistent sessions* or *session clustering*, is one method to share distributed session information among cluster members. 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 so it should be implemented in conjunction with hardware clustering products such as IBM HACMP, TSA, or solutions such as database replication. Another drawback is the performance hit, caused by database disk I/O operations and network communications.

Memory-to-memory session replication

Memory-to-memory replication enables the sharing of sessions between application servers without using a database. 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. The administrator can define how many replicas should exist in the domain (either a single replica, a defined number, or the entire domain).

Memory-to-memory replication, such as database persistence, also incurs a performance hit, primarily because of the overhead of network communications. Additionally, because 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: Depending on your application (for example, the session size) and on your hardware resources, memory-to-memory replication or database persistence might be the better solution for your environment. Refer to *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392 for additional information about resource requirements and performance for either option.

Conclusion

Storing session state 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 might 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.

You can find more information about this topic, including traces and logs, 2.5, “Web container clustering and failover” on page 51 and Chapter 6 of *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392.

2.4.2 EJB sessions or transactions

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

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.

Stateful session beans

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 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* to direct client requests. After an 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 the following caching options:

- 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.

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 starts 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.

2.4.3 Server affinity

The previous discussion 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 always directs a client's access of a stateful session bean to the single server instance that contains 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 either fails or that server itself is 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 might be expensive, and it might 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 might 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 might be acceptable targets for a given request, but it also recognizes that each request might have a particular affinity for being directed to a particular server where it can be handled better or faster.

We encounter this notion of server affinity throughout the discussion of the various load-distribution facilities. In particular, we 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 also encounter the notion of *transaction affinity*, in which the load distribution facility recognizes the existence of a transaction, and behaves similarly.

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 might not always be able to provide a total guarantee that this affinity is respected. In a strong affinity mechanism, the system guarantees that affinity is always strictly respected and generates an error when it cannot.

2.5 Web container clustering and failover

A complete WebSphere environment can include several Web server instances as well as several WebSphere Application Server instances. Each HTTP server is configured to run the WebSphere Web server plug-in. The cluster members can all reside on a single node or can be distributed across multiple nodes in the WebSphere cell (vertical or horizontal scaling).

Each request coming into the Web server is passed through the plug-in, which uses its configuration information to determine if the request should be routed to WebSphere, and if so, to which application server (that is to which Web container) the request should be routed to (see Figure 2-5 on page 52). The communication between the plug-in and the application servers can be either HTTP or HTTPS. The Web server plug-in distributes requests around cluster members that are not available.

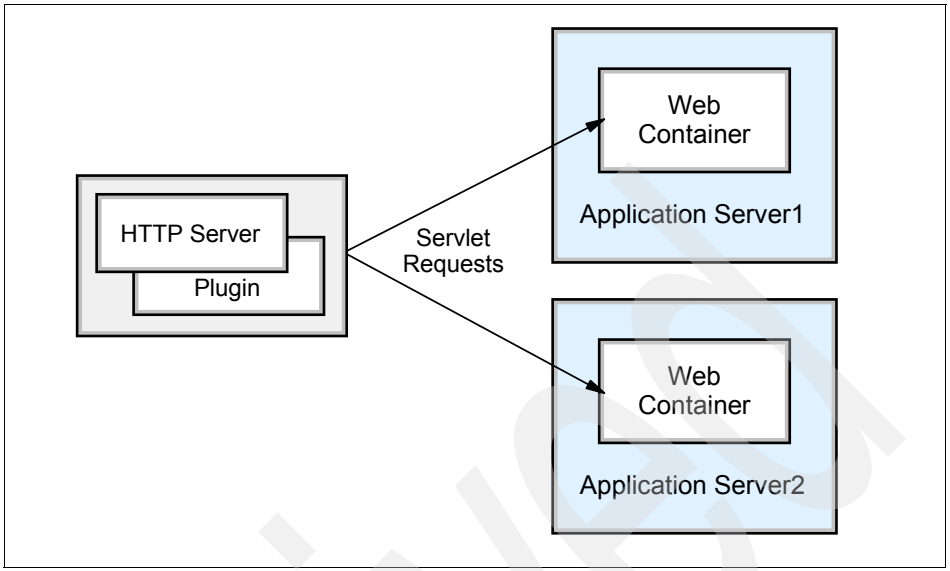


Figure 2-5 Plug-in (Web container) workload management

The plug-in, which runs in-process with the Web server itself, is responsible for deciding which Web container the request should be passed to. It uses the following mechanisms for WLM and failover:

- Application server clustering which creates server process redundancy for failover support. All application servers in a cluster host the same application or applications.
- The workload management routing technique built into the WebSphere Web server plug-in. It controls the routing of client requests among redundant server processes. 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 different, the plug-in routes requests to those cluster members with the higher weight value more often.
- Session management and failover mechanism, which provides HTTP session data for redundant server processes.

Thus, satisfactory failover support for Web clients can only be achieved by the use of all three mechanisms.

2.5.1 Session management and failover inside the plug-in

As you know, the plug-in always attempts to route a request that contains session information to the application server that processed the previous requests. However, if 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).

There are three methods of identifying a user's session to the application server: Cookies, URL rewriting, and SSL ID. Example 2-1 shows a JSESSIONID cookie which consists of four parts:

- Cache ID (0000)
- Session ID (A2MB4IJozU_VM8IffsMNfdR)
- Separator (:)
- Clone ID (v544d0o0 = application server ID)

Example 2-1 Example of a session identifier - JSESSIONID cookie

```
JSESSIONID=0000A2MB4IJozU_VM8IffsMNfdR:v544d0o0
```

In case of a failover, the Clone ID of the failover server is appended at the end, also separated by a colon. When the original server becomes available again, the request falls back and is handled by the original server.

Figure 2-6 on page 54 and the subsequent step-by-step explanation explain how the plug-in performs the failover.

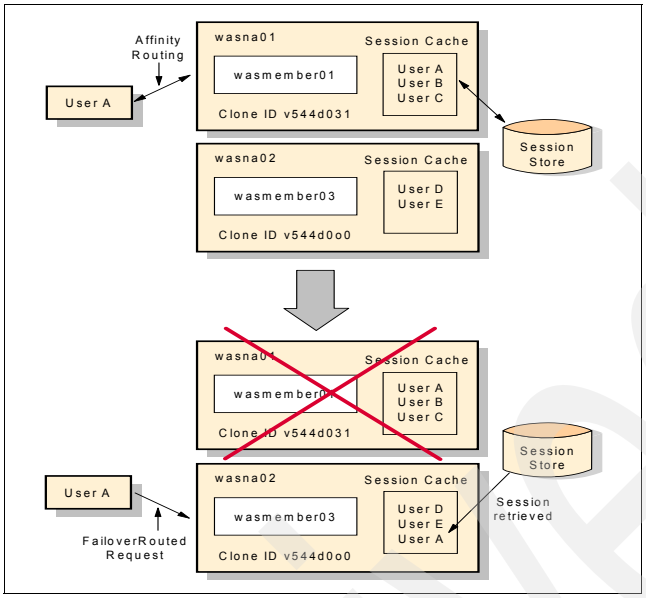


Figure 2-6 Session management example

Using Figure 2-6, the steps involved to find a failover application server are:

1. The plug-in processes a request from user A to `http://http1/snoop`. The request also contains a JSESSION cookie with a session ID and Clone ID of v544d031.
2. The plug-in matches the virtual host and URI to the cluster wascluster01 (composed by servers wasmember01 and wasmember03, each one located in a different machine).
3. The plug-in checks for session affinity and finds the Clone ID of v544d031 in the request's JSESSIONID cookie.
4. The plug-in searches for the Clone ID of v544d031 in the plug-cfg.xml's list of primary servers and matches the Clone ID to the wasmember01 application server.
5. The plug-in checks to see if wasmember01 has been marked down. In our case, it has not been marked down yet.
6. The plug-in attempts to get a stream to wasmember01. Finding the server is not responding, Web1 is marked as down and the retry timer is started.
7. The plug-in checks the session identifier again.
8. The plug-in checks the servers. When it reaches wasmember01, it finds it is marked down and the retry timer is not 0, so it skips wasmember01 and checks the next cluster member in the primary server list.

9. The plug-in selects wasmember03 (Clone ID v544d0o0) and 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 wasmember03 (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.

For additional information about plug-in failover behavior, read “WebSphere plug-in behavior” in Chapter 6 of *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392. This section discusses many failure situations in detail and includes information about logs and traces.

2.5.2 Web container failures

In a clustered environment with several cluster members, an unavailable application server does not mean an interruption of the service. When the plug-in has selected a cluster member to handle a request it will attempt to communicate with the cluster member. There are however a number of situations when the plug-in might not be able to complete a request to a specific application server. If this communication is unsuccessful or breaks, then the plug-in marks the cluster member as down and attempts to find another cluster member to handle the request. Web container failures are detected based on TCP response values or lack of response to a plug-in request.

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 following are some example scenarios when the plug-in cannot connect to a cluster member:

- Expected application server failures (the cluster member has been brought down intentionally for maintenance, for example).
- Unexpected server process failures (the application server JVM has crashed, for example).
- Server network problems between the plug-in and the cluster member (a router is broken, for example).
- System problems (whether expected), such as system shutdown or power failures.
- The cluster member is overloaded and cannot process the request (for example because the system is too small to handle a large number of clients, or because the server weight is inappropriate).

In the first two failure cases described, 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 might 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 58 for more information.

In the fifth case, overloading can make a healthy server unavailable. To avoid overloading of servers, you can define the maximum number of connections that are allowed from HTTP servers to the application server. This is explained in “Maximum number of connections” on page 61.

2.5.3 Web server plug-in failover tuning

The Web server plug-in uses an XML configuration file called plugin-cfg.xml to determine information about the WebSphere cell it is serving. 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. Plug-in file tags related to workload management and failover are:

- You can change the retry interval for connecting to a cluster member marked as down. See “Retry interval” on page 57 for more information.
- You can add the ConnectTimeout attribute for each server. See “Connection Timeout setting” on page 58.
- You can divide the servers into a primary server list and a backup server list. This is a feature available since WebSphere V5, also called two-level failover support. See “Primary and backup servers” on page 60 for information.
- 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. Refer to “Maximum number of connections” on page 61.

Retry interval

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 a server is unavailable, for example because it is being serviced. The default is 60 seconds. This default setting means that if a cluster member was marked as down, the plug-in would not retry it for 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.

This setting is specified in the configuration of each Web server in the Retry interval field. To change this value, go to **Servers** → **Web Servers** → ***WebServer_Name*** → **Plug-in properties** → **Request Routing.**

Finding the correct setting

There is no way to recommend one specific value; the value chosen depends entirely on your environment, for example on the number of cluster members in your configuration.

Setting the retry interval to a small value allows an application server that becomes available to quickly begin serving requests. However, too small of a value can cause serious performance degradation, or even cause your plug-in to appear to stop serving requests, particularly in a machine outage situation.

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 want your cluster members to be retried more often. 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 need a longer retry interval.

Another factor to consider for finding the correct retry interval for your environment is the operating system TCP/IP timeout value. To explain the relationship between these two values, let's look at an example configuration with two machines, which we call A and B. Each of these machines is running two clustered application servers (CM1 and CM2 on A, CM3 and CM4 on B). The HTTP server and plug-in are running on AIX with a TCP timeout of 75 seconds, the retry interval is set to 60 seconds, and the routing algorithm is weighted

round-robin. If machine A fails, either expected or unexpected, the following process occurs when a request comes in to the plug-in:

1. The plug-in accepts the request from the HTTP server and determines the server cluster.
2. The plug-in determines that the request should be routed to cluster member CM1 on system A.
3. The plug-in attempts to connect to CM1 on machine A. Because the physical machine is down, the plug-in waits 75 seconds for the operating system TCP/IP timeout interval before determining that CM1 is unavailable.
4. The plug-in attempts to route the same request to the next cluster member in its routing algorithm, CM2 on machine A. Because machine A is still down, the plug-in must again wait 75 seconds for the operating system TCP/IP timeout interval before determining that CM2 is also unavailable.
5. The plug-in attempts to route the same request to the next cluster member in its routing algorithm, CM3 on system B. This application server successfully returns a response to the client, about 150 seconds after the request was first submitted.
6. While the plug-in was waiting for the response from CM2 on system A, the 60-second retry interval for CM1 on system A expired, and the cluster member is added back into the routing algorithm. A new request is routed to this cluster member which is still unavailable, and this lengthy waiting process will begin again.

There are two options to avoid this problem:

- The recommended approach is to configure your application servers to use a non-blocking connection. This eliminates the impact of the operating system TCP/IP timeout. See “Connection Timeout setting” on page 58 for information.
- An alternative is to set the retry interval to a more conservative value than the default of 60 seconds, related to the number of cluster members in your configuration. A good starting point is 10 seconds + (#_of_cluster_members * TCP_Timeout). This ensures that the plug-in does not get stuck in a situation of constantly trying to route requests to the failed members. In the scenario described before, this setting would cause the two cluster members on system B to exclusively service requests for 235 seconds before the cluster members on system A are retried, resulting in another 150-second wait.

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 can the plug-in 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 failover 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 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 cannot connect.

To overcome this issue, WebSphere Application Server V6 offers an option within the plug-in configuration that allows you to change the connection timeout between the plug-in and each Application Server, which makes the plug-in use a non-blocking connect. 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.

Finding the correct setting

To determine what setting you should use, take into consideration how fast your network and servers are. You should test 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 marks it as down. This value is an application server property and thus can be set 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 see a slightly different trace output. Example 2-2 shows what you see in the plug-in trace if a non-blocking connect is successful.

Example 2-2 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
...
```

Primary and backup servers

Starting with V5, WebSphere Application Server 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 or session affinity. The Web server plug-in does not route requests to any server in the Backup Server list as long as there are application servers available from the Primary Server list. 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 does not apply any longer. To learn about partition ID, refer to *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392.

You can change a cluster member role (primary or backup) using the Administrative Console. Select **Servers** → **Application servers** → ***<AppServer_Name>*** → **Web Server plug-in properties** and select the appropriate value from the Server Role pull-down field.

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 2-3.

Example 2-3 ServerCluster element depicting primary and backup servers

```
...
<ServerCluster>
...
</Server>
   <PrimaryServers>
      <Server Name="wasna01_wasmember01"/>
      <Server Name="wasna02_wasmember03"/>
   </PrimaryServers>
   <BackupServers>
      <Server Name="wasna01_wasmember02"/>
      <Server Name="wasna02_wasmember04"/>
   </BackupServers>
</ServerCluster>
...
```

The backup server list is only used when all primary servers are down.

Maximum number of connections

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 that 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. The Maximum number of connections attribute is used to specify the maximum number of pending connections to an application server.

If there has been a successful connection but it is waiting for a thread in the Web container, then the plug-in waits for a response (and so does the client). If the connection backlog is full, the plug-in is refused a connection to the port and the plug-in basically treats this in the same way as a stopped cluster member. It

skips this cluster member and selects another cluster member instead. The cluster member then can reduce its connection backlog, because the plug-in does not send new requests. The plug-in checks whether the server is below the defined number of maximum connections periodically and only sends requests again when this status is reached, as shown in Example 2-4, Example 2-5 on page 63, and Example 2-6 on page 63.

Each application server can have a different setting for the maximum number of pending connections. To change this setting, go to **Servers** → **Application servers** → ***AppServer_Name*** → **Web Server plug-in properties**. The default setting is No Limit, which is the same as though the value is set to -1 or zero. You can set the attribute 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.

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), the plug-in log, the HTTP servers' access log, and Tivoli Performance Viewer.

For our test we have configured wasmember05 to have a maximum connections setting of 3 and wasmember06 of 4. We have one HTTP server fronting the application servers. Running a load test with 30 concurrent users against this cluster eventually results in both members having reached the maximum connections value. At this point, the user gets Error 503. See Example 2-4 (plug-in log) and Example 2-5 on page 63 (HTTP server access log).

Example 2-4 Plug-in log file errors when no server can serve requests (MaxConnections)

```
...
[Wed May 11 07:26:23 2005] 00007ab5 aaff9bb0 - STATS: ws_server_group:
serverGroupCheckServerStatus: Checking status of wasna01_wasmember05,
ignoreWeights 0, markedDown 0, retryNow 0, wlbAllows 17
reachedMaxConnectionsLimit 1
[Wed May 11 07:26:23 2005] 00007ab5 aaff9bb0 - WARNING: ws_server_group:
serverGroupCheckServerStatus: Server wasna01_wasmember05 has reached maximmum
connections and is not selected
[Wed May 11 07:26:23 2005] 00007ab5 aaff9bb0 - STATS: ws_server_group:
serverGroupCheckServerStatus: Checking status of wasna02_wasmember06,
ignoreWeights 0, markedDown 0, retryNow 0, wlbAllows 14
reachedMaxConnectionsLimit 1
```

```
[Wed May 11 07:26:23 2005] 00007ab5 aaff9bb0 - WARNING: ws_server_group:
serverGroupCheckServerStatus: Server wasna02_wasmember06 has reached maximmum
connections and is not selected
[Wed May 11 07:26:23 2005] 00007ab5 aaff9bb0 - ERROR: ws_server_group:
serverGroupNextRoundRobinServer: Failed to find a server; all could be down or
have reached the maximimum connections limit
[Wed May 11 07:26:23 2005] 00007ab5 aaff9bb0 - WARNING: ws_common:
websphereFindServer: Application servers have reached maximum connection limit
[Wed May 11 07:26:23 2005] 00007ab5 aaff9bb0 - ERROR: ws_common:
websphereWriteRequestReadResponse: Failed to find a server
[Wed May 11 07:26:23 2005] 00007ab5 aaff9bb0 - ERROR: ESI: getResponse: failed
to get response: rc = 8
[Wed May 11 07:26:23 2005] 00007ab5 aaff9bb0 - ERROR: ws_common:
websphereHandleRequest: Failed to handle request
...
```

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 2-5.

Example 2-5 HTTP Server access log example

```
[11/May/2005:07:26:23 -0500] "GET /wlm/BeenThere HTTP/1.1" 503 431
[11/May/2005:07:26:23 -0500] "GET /wlm/BeenThere HTTP/1.1" 503 431
```

Further down in the plug-in log you can see that eventually both servers respond to requests again when they have reduced their backlog. This is shown in Example 2-6.

Example 2-6 Max. connections - application servers pick up work again

```
...
#### wasmember05 worked off the pending requests and is "back in business",
wasmember06 is still in "reachedMaxConnectionsLimit" status:

[Wed May 11 07:26:23 2005] 00007ab9 b640bbb0 - STATS: ws_server:
serverSetFailoverStatus: Server wasna01_wasmember05 : pendingConnections 0
failedConnections 0 affinityConnections 0 totalConnections 1.
[Wed May 11 07:26:23 2005] 00007ab9 aa5f8bb0 - STATS: ws_server_group:
serverGroupCheckServerStatus: Checking status of wasna01_wasmember05,
ignoreWeights 0, markedDown 0, retryNow 0, wlbAllows 17
reachedMaxConnectionsLimit 0
[Wed May 11 07:26:23 2005] 00007ab9 aa5f8bb0 - STATS: ws_server:
serverSetFailoverStatus: Server wasna01_wasmember05 : pendingConnections 0
failedConnections 0 affinityConnections 0 totalConnections 2.
[Wed May 11 07:26:23 2005] 00007ab9 a9bf7bb0 - STATS: ws_server_group:
serverGroupCheckServerStatus: Checking status of wasna02_wasmember06,
```

```
ignoreWeights 0, markedDown 0, retryNow 0, wlbAllows 14
reachedMaxConnectionsLimit 1
[Wed May 11 07:26:23 2005] 00007ab9 a9bf7bb0 - WARNING: ws_server_group:
serverGroupCheckServerStatus: Server wasna02_wasmember06 has reached maximmum
connections and is not selected
...
#### wasmember06 is working off the the requests. Once it reduced the number of
pending connections to below 4 (which is the maximum) it can then also serve
requests again. Both servers are handling user requests now:
...
[Wed May 11 07:26:23 2005] 00007ab5 b4608bb0 - STATS: ws_server:
serverSetFailoverStatus: Server wasna02_wasmember06 : pendingConnections 3
failedConnections 0 affinityConnections 0 totalConnections 4.
[Wed May 11 07:26:23 2005] 00007ab9 b5009bb0 - STATS: ws_server:
serverSetFailoverStatus: Server wasna02_wasmember06 : pendingConnections 2
failedConnections 0 affinityConnections 0 totalConnections 4.
[Wed May 11 07:26:23 2005] 00007ab9 b3c07bb0 - STATS: ws_server:
serverSetFailoverStatus: Server wasna02_wasmember06 : pendingConnections 1
failedConnections 0 affinityConnections 0 totalConnections 4.
[Wed May 11 07:26:23 2005] 00007ab9 b5a0abb0 - STATS: ws_server:
serverSetFailoverStatus: Server wasna02_wasmember06 : pendingConnections 0
failedConnections 0 affinityConnections 0 totalConnections 4.
[Wed May 11 07:26:24 2005] 00007ab9 b3206bb0 - STATS: ws_server_group:
serverGroupCheckServerStatus: Checking status of wasna02_wasmember06,
ignoreWeights 0, markedDown 0, retryNow 0, wlbAllows 14
reachedMaxConnectionsLimit 0
...
```

This feature helps you to better load balance the application servers fronted by the plug-in. If application servers are overloaded, the plug-in skips these application servers automatically and tries the next available application server.

However, a better solution is to have an environment that can handle the load that you are expecting and to have it configured correctly. This includes setting weights that correspond to the system capabilities, having the correct balance of cluster members and Web servers, and setting up the queues for requests and connections.

2.6 EJB container clustering 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.

EJB clients can be servlets, JSPs, a J2EE client, stand-alone Java applications, or other EJBs. 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).

This section gives an overview and introduction into EJB WLM. For information about Enterprise Java Services workload management, see Chapter 7 “EJB workload management” of *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392.

In WebSphere Application Server V6, the Deployment Manager is not a single point of failure for WLM routing. With the new High Availability Manager (HAManager), a failure of the Deployment Manager triggers the HA Coordinator, which carries the WLM routing information, to failover 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. See Chapter 6, “WebSphere HAManager” on page 175 for more information.

The option to configure Backup clusters further enhances EJB application availability. See 2.7, “Backup cluster support” on page 87 for more information.

2.6.1 EJB container redundancy

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. Figure 2-7 shows horizontal and vertical scaling that is used for application server redundancy to tolerate possible process and machine failures.

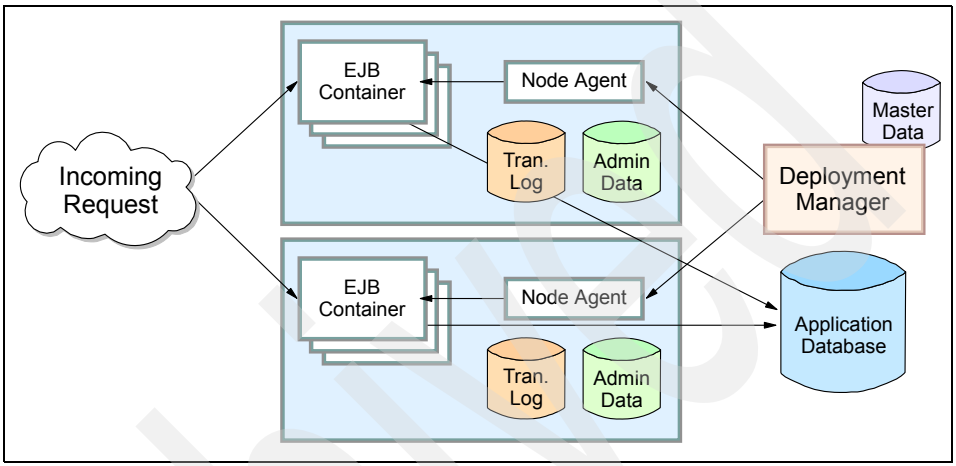


Figure 2-7 WebSphere EJB container failover

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.

2.6.2 EJB bootstrapping considerations

In order to access EJBs that are 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 InititalContext, 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. 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.

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.

It is very important that you provide fault-tolerant bootstrapping information to the EJB client to allow for application server (EJB container) failures. If the bootstrap server and port are not available, then the reference cannot be obtained. Therefore, it is highly recommended that you do not use the Deployment Manager as the bootstrapping server as the Deployment Manager can be a single point of failure in your WebSphere environment. Instead, use multiple application servers or Node Agents.

Detailed information about EJB bootstrapping can be found in Chapter 7 “EJB workload management” of the redbook *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392. This chapter contains code samples and lists best practices for EJB bootstrapping.

2.6.3 EJB client redundancy and bootstrap failover support

When planning an EJB HA solution, in addition to EJB server redundancy, you should also consider EJB client failover and redundancy. EJB client redundancy refers to the automatic failover capability for an EJB request originator. In other words, a 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). You should consider the following scenarios:

- 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 might 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, because the client is not redundant. You should bootstrap the client to as many bootstrap servers as possible, as shown in Example 2-7.

Example 2-7 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 this example, the EJB client has the information for two bootstrap servers. Therefore, if the request to server host1 fails, the bootstrap engine redirects the bootstrap request to the server on host2 automatically.

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, because the InitialContext is workload-managed and you can use the simple name in the lookup. The

InitialContext being workload-managed means that all the cluster members in the cluster are available in the InitialContext object. This allows the WLM plug-in to send the requests to another server in the cluster if a request to one server fails. 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. This means that if the Node Agent that returns the InitialContext fails then the InitialContext is invalid and will not work, even if you use multiple Node Agents on the CorbaLoc URL. In this option, the Node Agent is a single point of failure.

For J2EE clients, you can specify a bootstrap host and port in the launch command line, and try different hosts or ports if you do not succeed when using the default URL provider.

When the EJB home is location successfully, 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.

2.6.4 EJB types, workload management and failover

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 plug-in. EJB failover depends on whether this type of EJB can be workload-managed by the container.

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

The only type of EJB references not subject to load distribution through EJB WLM are stateful session bean instances. However, IBM WebSphere Application Server Network Deployment V6 now supports stateful session bean failover as explained in 2.6.5, “Stateful session bean failover” on page 73.

Table 2-1 summarizes the workload management capability of different types of EJBs.

Table 2-1 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
Session Bean	Home	Yes
	Stateless bean instance	Yes
	Stateful bean instance	No

Stateless session beans

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.

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.

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 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.

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.

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 workload management. This is a new feature in WebSphere V6 and is discussed in more detail in 2.6.5, “Stateful session bean failover” on page 73.

Unlike the failover support for stateless session beans, the highly available stateful session bean does not use a redundant array of stateful session bean instances. Rather, it 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.

Entity beans

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 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. See “Entity beans” on page 48 for a detailed description of the three caching options.

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 might 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, go to the InfoCenter and search for *concurrency control*.

EJB failover capabilities summary

With the addition of the new stateful session bean failover, IBM WebSphere Application Server Network Deployment V6 supports failover for almost all types of Enterprise JavaBeans. Table 2-2 summarizes the failover capability for the various EJB types.

Table 2-2 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

2.6.5 Stateful session bean failover

One significant high availability 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 using the functions of the Data Replication Service (DRS) and server workload management (WLM). Because it is a new feature, we describe how to configure stateful session bean failover as well as best practices in this section.

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 users who want to enable failover under the following circumstances:

- 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.
- 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.
- 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.
- 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.

Enable or disable stateful session bean failover

Now, let us look at how to enable the failover support for stateful session beans at the three different levels.

At EJB container level

To enable stateful session bean failover at the EJB container level, do the following:

1. In the Administrative Console, select **Servers** → **Application servers** → ***<AppServer_Name>***.
2. Expand **EJB Container Settings**, and then select **EJB container**.
3. Select **Enable stateful session bean failover using memory-to-memory replication**, as shown in Figure 2-8.

   This option 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.

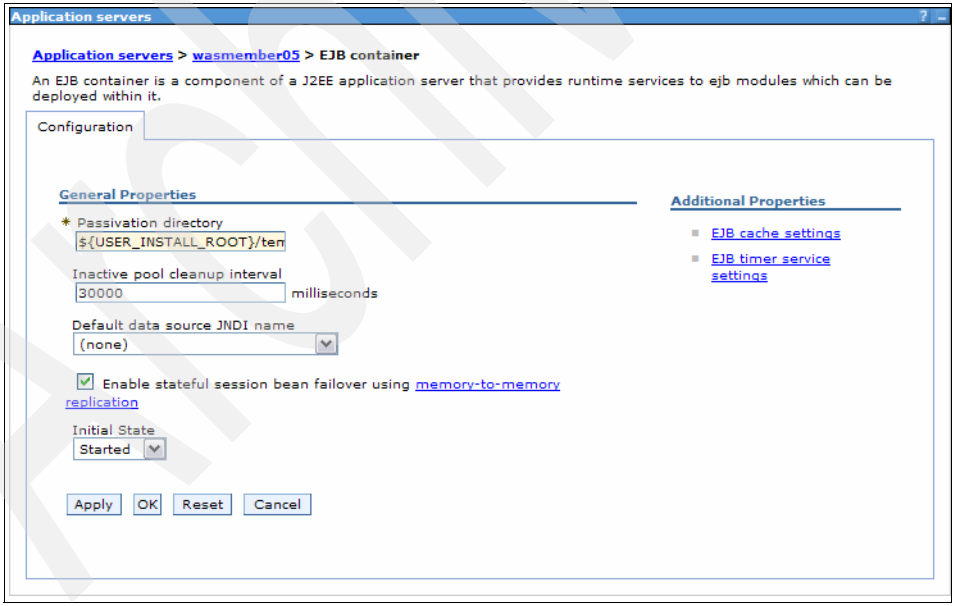


Figure 2-8 Configure stateful session bean failover at EJB container level

At application level

To enable stateful session bean failover at the application level, do the following:

1. In the Administrative Console, select **Applications** → **Enterprise Applications** → ***<Application_name>***.
2. Select **Stateful session bean failover settings** from the Additional Properties.
3. Select **Enable stateful session bean failover using memory to memory replication**, which enables failover for all stateful session beans in this application. If you want to *disable* the failover, deselect this option.
4. Define the Replication settings as shown in Figure 2-9 on page 76. You have two choices:
   - Use replication settings from EJB container

     If you select this option, any replication settings that you define for this application are ignored (that is, you do not overwrite the EJB container settings).
   - Use application replication settings

     Selecting this option overrides the EJB container settings. This option 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.
5. Click **OK** and save your changes.

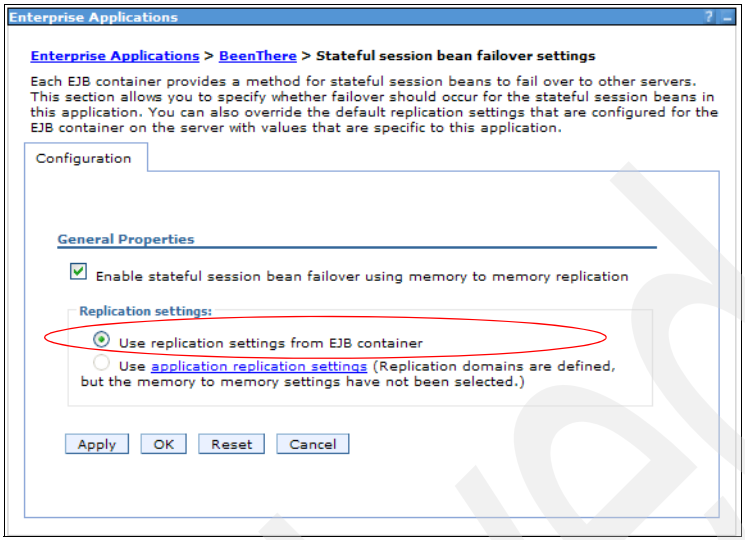


Figure 2-9 Configure stateful session bean failover at application level

At EJB module level

To enable stateful session bean failover at the EJB module level, do the following:

1. In the Administrative Console, select **Applications** → **Enterprise Applications** → ***<Application_name>***.
2. Under Related Items, select **EJB Modules**.
3. Select the .jar file with which you want to work.
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 2-10 on page 77. You have two choices:
   - Use application or EJB container replication settings

     If you select this option, any replication settings that are defined for this EJB module are ignored.
   - Use EJB module replication settings

     Selecting this option overrides the replication settings for the EJB container and application. This option 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.

Attention: If you use application or EJB container replication settings, then memory-to-memory replication must be configured at the EJB container level. Otherwise, the EJB ignores settings on this panel during server startup, and the EJB container logs a message indicating that stateful session bean failover is not enabled for this application.

7. Select **OK** and save your changes.

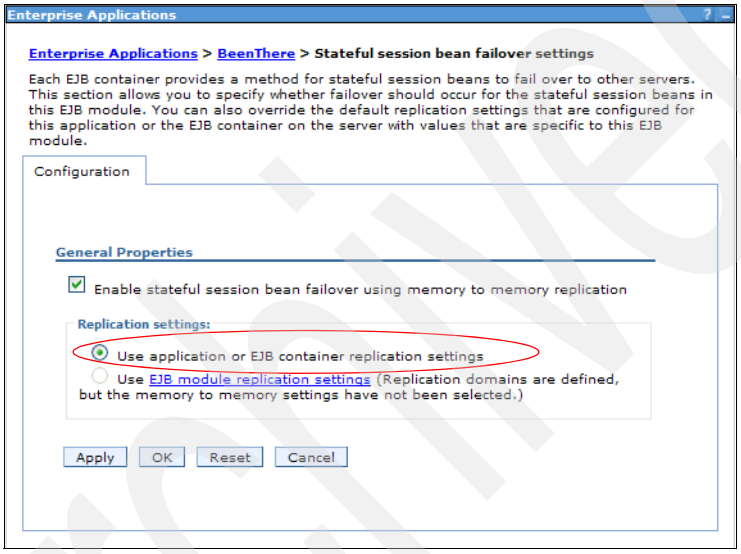


Figure 2-10 Configure stateful session bean failover at EJB module level

Stateful session bean failover best practices

When designing and applying the failover support for stateful session beans on your applications, consider the following best practices:

- If the stateful session bean is 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 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 might 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.

  Attention: To avoid this situation, remember that with failover enabled, your application should never use both a local (EJBLocalObject) and remote (EJBObject) references to the same stateful session bean instance.

Example: stateful session bean failover

The best way to obtain what happened during a failover on the different application servers is to check their logs. First, take a look at the logs of the failed server, which should show the restarting process initiated by the Node Agent (if the Node Agent did not fail also). Then look at the application server logs that took over the failed stateful session beans.

Our sample configuration is very simple. We have one cell, known as Cell01, two nodes called Node01 and Node02, a cluster called Cluster01, and its two cluster members, FailingServer and UpServer.

The log of the failed application server

For our example, we stopped application server FailingServer using a **`kill -9`** command to simulate a JVM crash. We then restarted the server automatically by the Node Agent, based on the Monitoring Policy settings for the application server. You first see the automatic restart of the server by the Node Agent, and then the server is up again. After that, you can see the resynchronization of the replication system. The following list steps you through this process:

- Automatic restart

  Example 2-8 on page 79 shows the beginning of the restart. You cannot see the stop of the server in the log because it was killed by the -9 signal. However, you can guess that it was a violent stop because there is no message that indicates the stop before the restarting messages. An important message that you should look for is `The startup trace state is *=info`.

Example 2-8 Restart of the failed node

```
************ Start Display Current Environment ************
WebSphere Platform 6.0 [ND 6.0.2.0 o0516.14]  running with process name
Cell01\Node01\FailingServer and process id 23912
...
Java Library path =
/opt/WebSphere/AppServer/java/bin/../jre/bin:/opt/WebSphere/AppServer/java/jre/
bin/classic:/opt/WebSphere/AppServer/java/jre/bin:/opt/WebSphere/AppServer/bin:
/opt/mqm/java/lib:/opt/wemps/lib:/usr/lib
************* End Display Current Environment *************
[4/26/05 16:26:06:274 CDT] 0000000a ManagerAdmin  I   TRAS0017I: The startup
trace state is *=info.
```

- The application server is ready again

  Example 2-9 shows the message `open for e-business`, which means the application server startup has completed.

> **Attention:** When testing the failover mechanism, you must wait for the message `REPLICATION_UP` that is shown in Example 2-10. It most probably appears *after* the `open for e-business` message. However, the replication is only available again after the `REPLICATION_UP` message. If you stop another application server before the replication is fully re-enabled, you might loose some information that is not replicated yet.

Example 2-9 Open for e-business message

```
[4/26/05 16:27:20:192 CDT] 00000020 WebContainer  A   SRVE0161I: IBM WebSphere
Application Server - Web Container.  Copyright IBM Corp. 1998-2004
[4/26/05 16:27:20:194 CDT] 00000020 WebContainer  A   SRVE0162I: Servlet
Specification Level: 2.4
[4/26/05 16:27:20:196 CDT] 00000020 WebContainer  A   SRVE0163I: Supported JSP
Specification Level: 2.0
[4/26/05 16:27:20:248 CDT] 00000020 WebContainer  A   SRVE0239I: Extension
Factory [class com.ibm.ws.webcontainer.extension.ExtHandshakeVHostExtension
Factory] was registered successfully.
[4/26/05 16:27:20:249 CDT] 00000020 WebContainer  A   SRVE0240I: Extension
Factory [class com.ibm.ws.webcontainer.extension.ExtHandshakeVHostExtension
Factory] has been associated with patterns [VH:_WS_EH* ].
...
[4/26/05 16:27:21:788 CDT] 00000020 SchedulerServ I   SCHD0078I: The Scheduler
Service has completed starting the Schedulers.
[4/26/05 16:27:21:803 CDT] 0000000a RMIConnectorC A   ADMC0026I: The RMI
Connector is available at port 9809
[4/26/05 16:27:22:168 CDT] 0000000a WsServerImpl  A   WSVR0001I: Server
FailingServer open for e-business
```

- The replication is up again

  Example 2-10 shows the REPLICATION_UP message, which indicates the successful restart of the replication service. This message might appear after the server is up.

Example 2-10 Event REPLICATION_UP

```
[4/26/05 16:27:35:129 CDT] 0000001d RoleViewLeade I   DCSV8054I: DCS Stack
DefaultCoreGroup.tri-store at Member Cell01\Node01\FailingServer: View change
in process.
[4/26/05 16:27:35:134 CDT] 00000014 VSync         I   DCSV2004I: DCS Stack
DefaultCoreGroup.tri-store at Member Cell01\Node01\FailingServer: The
synchronization procedure completed successfully. The View Identifier is
(1:0.Cell01\Node01\FailingServer). The internal details are [2].
[4/26/05 16:27:35:146 CDT] 00000014 ViewReceiver  I   DCSV1033I: DCS Stack
DefaultCoreGroup.tri-store at Member Cell01\Node01\FailingServer: Confirmed all
new view members in view identifier (4:0.Cell01\Node02\UpServer). View channel
type is View Ptp.
[4/26/05 16:27:35:149 CDT] 0000001d DataStackMemb I   DCSV8050I: DCS Stack
DefaultCoreGroup.tri-store at Member Cell01\Node01\FailingServer: New view
installed, identifier (4:0.Cell01\Node02\UpServer), view size is 2 (AV=2, CD=2,
CN=2, DF=2)
[4/26/05 16:27:39:642 CDT] 00000035 DRSBootstrapM A   CWWDR0001I:  Replication
instance launched : HttpSessionCache .
[4/26/05 16:27:39:648 CDT] 00000017 DRSBootstrapM A   CWWDR0001I:  Replication
instance launched : __homeOfHomes .
[4/26/05 16:27:39:645 CDT] 00000016 DRSBootstrapM A   CWWDR0001I:  Replication
instance launched : tri-ear .
[4/26/05 16:27:39:653 CDT] 00000035 SessionContex I   Received event
REPLICATION_UP.
```

The logs of the takeover server

This section discusses the most interesting logs. We captured these logs while stopping the other application server. You can first see that a server is no longer available, then the takeover, and finally that the failed server is up again.

- A server failed

  You can see the failure of the other server which induced a REPLICATION_DOWN event in Example 2-11. You do not see a message that indicates right away that another server has failed. Instead, the REPLICATION_DOWN event indicates that the replication domain lacks one of its members.

Example 2-11 Another server failed

```
[4/26/05 16:25:26:665 CDT] 0000001d SessionContex I   Received event
REPLICATION_DOWN.
```

- Taking over

  The REPLICATION_DOWN event triggers the recovery process. The failed server is known as FailingServer. Example 2-12 shows the DCS messages that tell you that the recovery processing is started and that FailingServer has been removed.

  The transaction recovery process ends with the status message that indicates how many transactions are recovered (Transaction service recovering 0 transactions).

Example 2-12 Recovering the failed server

```
[4/26/05 16:25:26:672 CDT] 00000042 RecoveryDirec I   CWRLS0011I: Performing
recovery processing for a peer WebSphere server (Cell01\Node01\FailingServer).
[4/26/05 16:25:26:673 CDT] 0000001e RoleMember    I   DCSV8052I: DCS Stack
DefaultCoreGroup.tri-store at Member Cell01\Node02\UpServer: Defined set
changed. Removed: [Cell01\Node01\FailingServer].
[4/26/05 16:25:26:679 CDT] 00000042 RecoveryDirec I   CWRLS0013I: All
persistent services have been directed to perform recovery processing for a
peer WebSphere server (Cell01\Node01\FailingServer).
[4/26/05 16:25:26:959 CDT] 00000042 RecoveryDirec I   CWRLS0013I: All
persistent services have been directed to perform recovery processing for a
peer WebSphere server (Cell01\Node01\FailingServer).
[4/26/05 16:25:27:112 CDT] 00000043 RecoveryManag A   WTRN0028I: Transaction
service recovering 0 transactions.
```

- The failed server comes up again

  Next, you see the failed server coming up again. The first action is to halt recovery processing for this server. In Example 2-13, the most interesting message is that the replication is up again: Received event REPLICATION_UP. That message means you are now back to the normal situation.

Example 2-13 The failed server is up again

```
[4/26/05 16:26:47:033 CDT] 00000014 ViewReceiver  I   DCSV1033I: DCS Stack
DefaultCoreGroup at Member Cell01\Node02\UpServer: Confirmed all new view
members in view identifier (7:0.Cell01\CellManager01\dmgr). View channel type
is View|Ptp.
[4/26/05 16:26:47:298 CDT] 0000001d RecoveryDirec I   CWRLS0014I: Halting any
current recovery processing for a peer WebSphere server
(Cell01\Node01\FailingServer).
[4/26/05 16:26:47:300 CDT] 0000001e RoleMember    I   DCSV8051I: DCS Stack
DefaultCoreGroup.tri-store at Member Cell01\Node02\UpServer: Defined set
changed. Added: [Cell01\Node01\FailingServer].
[4/26/05 16:26:47:378 CDT] 00000014 MbuRmmAdapter I   DCSV1032I: DCS Stack
DefaultCoreGroup.tri-store at Member Cell01\Node02\UpServer: Connected a
defined member Cell01\Node01\FailingServer.
```

```
[4/26/05 16:27:04:386 CDT] 0000001e RoleMergeLead I   DCSV8054I: DCS Stack
DefaultCoreGroup.tri-store at Member Cell01\Node02\UpServer: View change in
process.
[4/26/05 16:27:04:390 CDT] 00000014 VSync         I   DCSV2004I: DCS Stack
DefaultCoreGroup.tri-store at Member Cell01\Node02\UpServer: The
synchronization procedure completed successfully. The View Identifier is
(3:0.Cell01\Node02\UpServer). The internal details are [4].
[4/26/05 16:27:04:401 CDT] 0000001e DataStackMemb I   DCSV8050I: DCS Stack
DefaultCoreGroup.tri-store at Member Cell01\Node02\UpServer: New view
installed, identifier (4:0.Cell01\Node02\UpServer), view size is 2 (AV=2, CD=2,
CN=2, DF=2)
[4/26/05 16:27:04:404 CDT] 00000014 ViewReceiver  I   DCSV1033I: DCS Stack
DefaultCoreGroup.tri-store at Member Cell01\Node02\UpServer: Confirmed all new
view members in view identifier (4:0.Cell01\Node02\UpServer). View channel type
is View|Ptp.
[4/26/05 16:27:04:937 CDT] 00000018 SessionContex I   Received event
REPLICATION_UP.
```

2.6.6 WebSphere process failures, relationship to EJB processing

For a specific WebSphere environment, the following three types of servers can fail:

- Deployment Manager
- Node Agent
- WebSphere Application Server cluster member

Each process failure has a different effect on EJB processing.

Deployment Manager failure

Prior to IBM WebSphere Application Server Network Deployment V6, the Deployment Manager provided the runtime support for WLM services, which is the key to a successful EJB failover. Thus, when the Deployment Manager failed, the WLM information was no longer propagated to associated cluster members and EJB clients. This issue presented a single point of failure for EJB workload management.

IBM WebSphere Application Server Network Deployment V6 eliminates this issue with the introduction of the WebSphere High Availability service. The HAManager is now responsible for running key services such as WLM on all available servers rather than on a single dedicated Deployment Manager. When 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 delegation provides continuous service for EJB workload management. Therefore, the Deployment Manager is no longer a

single point of failure for EJB workload management, which improves WebSphere high availability.

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 failover to another LSD. Because 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 Chapter 7 of *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392 for detailed 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 which to route the request.

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 might lead to a timeout. Under these circumstances, it might be helpful to change the server weight 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 or individual machine errors) before a connection to a server has been established, the operating system TCP/IP keep-alive timeout dominates the behavior of the system's response to a request. This is because a client waits for the OS-specific timeout before a failure is detected.
   - 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 timeouts repeatedly. Great care must be taken to tune it properly. If the value is set too high, failover can become very slow; if it is set too low, requests might time out before the server has a chance to respond.

     The two most critical factors affecting the choice of a timeout 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 can use a smaller timeout 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 timeout 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 the following as a command-line argument for the client process:

```
-Dcom.ibm.websphere.wlm.unusable.interval=<seconds>
```

2.6.7 EJB WLM exceptions

There are two possible scenarios:

- A failure that occurs during EJB processing

  This triggers an exception. Normally, WebSphere WLM catches the exception and resends the failed request to another application server in the cluster. This is the normal failover functionality of WebSphere EJB WLM. If, however, WLM cannot determine whether the transaction completed, it sends an exception to the application.

- A failure that occurs during WLM processing

  If the WLM mechanism is not able to handle the request it sends an exception to the application.

Exceptions that cause WLM to failover

The EJB workload management catches most of the exceptions that can occur during execution. WLM handles these exceptions and decides whether the original request should be redirected to another available cluster member. These exceptions are:

- org.omg.CORBA.COMM_FAILURE
- org.omg.CORBA_NO_RESPONSE

These exceptions have a COMPLETION_STATUS of NO, YES, or MAYBE and this value actually determines whether the request fails over to another available member:

- With a COMPLETION_STATUS of COMPLETED_NO automatic failover occurs because the request was not completed. The request is then rerouted to an available cluster member.

- With a COMPLETION_STATUS of COMPLETED_YES, there is no need to failover because the transaction was successfully completed. Some communication errors might have occurred during the marshalling of the answer.
- With a COMPLETION_STATUS of COMPLETED_MAYBE, WLM cannot verify whether the transaction was completed and thus cannot redirect the request. WLM, according to the programming model, sends this exception to the client application. It is the applications' responsibility to handle this exception and to decide whether to retry the request.

Important: This exception must be handled by the application. It does not mean that something is broken. The application has to check whether the transaction was successful and, depending on the result, should then issue the request. See also "Something unusual happened, check and retry if necessary" on page 87.

Exceptions during failover

During the failover process, WLM might face problems and then either tries to redirect the request again or issues a CORBA exception. This means that something unusual occurred while WLM was trying to redirect the original request to another server. There are two reasons for such an exception to occur. Both events are contained in a org.omg.CORBA.TRANSIENT exception, with different minors:

- Communication problem with one of the other servers

  TRANSIENT_SIGNAL_RETRY (minor=1229066306), meaning that an error occurred during the communication and that the client should retry the request.

- No reachable server

  NO_IMPLEMENT_NO_USEABLE_TARGET (minor=1229066304), meaning that WLM did not find any cluster member able to answer the request. This is the worst case because the application cannot retry the request because it would fail again. This should then lead to a message to the user, for example, asking to contact the system administrator.

EJB WLM exceptions: the application design point of view

In situations where EJB WLM is not able to failover the request to another server or does not know the status of a request, the application developer must ensure that exceptions that are sent to the application are handled correctly. The application should handle three possible return status after calling an EJB: OK, retry, or error:

- Everything is OK, continue

  This is the way it should always be! No further action is required.

- Something unusual happened, check and retry if necessary

  WLM tried to complete the request but an unexpected event occurred and WLM cannot re-issue the request on its own. In this case, the application should check and retry the request if needed. This behavior should be implemented if a CORBA exception was caught at the application level with the following:

  - COMPLETION_STATUS of COMPLETED_MAYBE
  - Minor code of TRANSIENT_SIGNAL_RETRY (minor=1229066306)

- Unrecoverable error

  The most likely cause of such an error is the unavailability of any cluster member so the request cannot failover to another server. The application will detect such a situation when it catches a CORBA exception with a minor code of NO_IMPLEMENT_NO_USEABLE_TARGET (minor=1229066304). In this case, an error message should be issued to the user.

2.7 Backup cluster support

IBM WebSphere Application Server Network Deployment V6 supports a mirrored backup cluster that can failover EJB requests from a primary cluster in one cell to the mirrored backup cluster in another cell when the primary cluster fails. Fail back is automatic as soon as the primary cluster becomes available.

There are certain situations in which you would use backup cluster support instead of other high availability options. Any scenario must deal with IIOP traffic which could be from servlets, Java applications, the client container or other traffic. If your configuration contains two cells with mirrored applications, you might want to enable backup clusters for the case of catastrophic failure when an entire cell goes down.

Before you can enable backup cluster support, you need to create the backup cluster in another cell with the same cluster name as the primary cluster and deploy the same applications into both the primary cluster and the mirrored backup cluster, with the same resources in the backup cluster as in the primary

cluster. The primary cluster and the backup cluster must reside in separate cells. The backup cluster bootstrap host and port determine which cell contains the backup cluster.

The basic behavior for backup clusters requires that the Deployment Managers are running in order to perform the failover and possible failback. In WebSphere V6, backup cluster communication no longer only flows through the Deployment Manager. Although the Deployment Manager is still required for the failover (as it stores the location of the backup cluster), core groups and core group bridges allow backup cluster communication to be dispersed over multiple bridges.

The core group bridge service can be configured to allow communication between the core groups. The core group bridge service handles the communication between a primary and backup cluster. You can configure core group bridges between the Deployment Managers, Node Agents and servers to prevent a single point of communication failure. Each cluster can only have one backup cluster. The primary and backup clusters are configured to point to each other to allow fail back.

2.7.1 Runtime behavior of backup clusters

After a failover, if cluster members in the primary cluster become available again, the mirrored cluster support attempts to fail back the requests to the primary cluster. This fail back is automatic. However, for the failback to work, the primary cluster must be defined as a backup for the backup cluster. In other words, both primary and backup clusters must have a backup configured, and each cluster's backup must point to the opposite cluster.

Note that the mirrored cluster failover support is not a cell level failover or Node Agent level failover. For the most part, the mirrored cluster support depends on a running Deployment Manager and is limited to cluster failover only. For example, if the primary cluster's entire cell stops processing, a new client's requests to the primary cluster will fail and is not sent to the backup cluster. This is because information regarding the backup cluster cannot be retrieved from the primary cluster's cell because the primary cluster's cell is not processing. On the other hand, if the primary cluster's cell Deployment Manager, Node Agents, and application servers stop processing *after* the client has already been sending requests to the primary cluster, the client already knows about the backup cluster and is able to send the requests to the backup cluster.

2.7.2 Scenario and configuration description

The major steps to configure backup clusters are adding the backup cluster settings and optionally creating a core group bridge. Items to configure are:

1. WebSphere cell and cluster setup.
2. Security considerations. You need to import or export LTPA keys to handle security across cells.
3. Backup cluster configuration.
4. Core group bridge configuration (optional).

We use the IBM Trade Performance Benchmark Sample for WebSphere Application Server (called Trade 6 throughout this book) as the sample application in our backup cluster configuration. You can download Trade 6 at:

`http://www.ibm.com/software/webservers/appserv/was/performance.html`

Table 2-3 lists the host names, node names, and ports that we used in our sample configuration. Our domain name is ibmredbook.com.

Table 2-3 Backup cluster sample configuration information

Cell name	Deployment Manager information	Node Agent information	Cluster names
wascell02 (Primary cell)	► washost01 ► wasdmgr02 ► 9809 (Bootstrap) ► 9352 (DCS)	► washost02 ► wasna01 ► 9353 (DCS)	► wascluster01 (EJB modules) ► wascluster02 (Web modules)
wascell03 (Backup cell)	► washost02 ► wasdmgr03 ► 9810 (Bootstrap) ► 9353 (DCS)	► washost04 ► wasna01 ► 9354 (DCS)	► wascluster01 (EJB modules) ► wascluster02 (Web modules)

2.8 WebSphere cell and cluster setup

This section describes the setup of the WebSphere cluster. Although it is not recommended for performance reasons to separate the Web and EJB containers, this is necessary for the example to work (split-JVM environment). Therefore, we deploy the Web and EJB modules of Trade 6 into different clusters. The required steps are:

1. Create two cells with at least one node in each.
   - Cell names: wascell02 and wascell03
2. Create a cluster in each cell. Both clusters have to have the same name. If you install an application that splits across multiple clusters for the Web and

EJB modules (as is the case in this example), create the additional clusters as well.

- Trade 6 EJB modules cluster: wascluster01
- Trade 6 Web modules cluster: wascluster02

Note that you could also create the cluster and cluster members using the trade.jacl script used to set up the Trade 6 resources in the next step. However, with the split-JVM environment we are setting up, it is easier to create the clusters using the Administrative Console. The trade.jacl script asks for cluster and cluster member names and only creates a new cluster and cluster members if they do not exist in the cell yet.

3. Set up the Trade 6 resources by running the trade.jacl script with the **`configure`** option. Run the script on the primary and backup cell.

   Note: Follow the directions that are included in the Trade 6 download package for setup and installation, or see Chapter 8 of *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392 for instructions on how to use the trade.jacl script.

4. Install Trade 6 using the Administrative Console. Consider this:

   a. On the Map modules to servers step (Step 2), map the TradeEJB module to wascluster01 and map TradeWeb to wascluster02.

   b. Make sure the right database is selected on the Provide options to perform the EJB Deploy step (Step 3).

   c. On the Map EJB references to beans step (Step 9), update the EJB references. This enables them to contact the EJBs installed on the EJB cluster. You must change the JNDI names of the ejb/Trade reference bindings in the TradeWeb module to their fully qualified JNDI names, for example:

      cell/clusters/<EJB_Cluster_Name>/ejb/TradeEJB

      In our environment, the correct fully qualified JNDI name is:

      cell/clusters/wascluster01/ejb/TradeEJB

      So you must add cell/clusters/wascluster01/ to the beginning of the existing names for the ejb/Trade, ejb/Quote, ejb/LocalQuote, and ejb/LocalAccountHome settings.

2.8.1 Security considerations

If your application requires security between the Web modules and EJB modules, you need to configure security to work correctly between the cells by doing the following:

1. Review the time, date, and time zone on all machines in both cells. The machines in the primary and backup cells need to be within five minutes of each other.
2. Configure a user registry with an LDAP server, OS security or a custom user registry.
3. On the primary cell, go to **Security** → **Global security**. Under **Authentication,** select **Authentication mechanisms** → **LTPA.** Enter an LTPA password on the LTPA authentication page. Remember the password for later to use on the backup cluster. Click **Apply**, then save your changes.
4. Still on the LTPA authentication page, enter a path and file name in the Key file name field and click **Export Keys**. This exports the LTPA authentication keys. Later, they are imported into the backup cell.

   Key file name: `WebSphereKeys/primarycell.keys`

   Transfer the LTPA key file to the backup cluster's Deployment Manager's machine.
5. Back in the Administrative Console, still on the LTPA authentication page, under **Additional Properties**, select **Single signon (SSO)**. Enter an appropriate Domain name to include both cells.

   Domain name: `ibmredbook.com`
6. Configure any additional necessary security items.
7. On the Global security page, enable global security, and save and synchronize the changes. Make sure that all Node Agents are up and running when enabling global security.
8. Restart the primary cell.
9. On the backup cell, configure the same user registry as on the primary cell.
10. In the Administrative Console, go to **Security** → **Global security**. Under **Authentication**, select **Authentication mechanisms** → **LTPA.** In the Password and Confirm password fields, enter the password from the primary cell. Click **Apply** and save your changes.
11. Enter the path and file name of the exported LTPA key file from the primary cell into the Key file name field. Click **Import Keys**.

    If the password is incorrect, you see an error message in the WebSphere Administrative Console. Correct the password, click apply, and then repeat this step.

12. On the same panel, under **Additional Properties**, select **Single signon (SSO)**. Enter an appropriate Domain name to include both cells. It should be the same as the primary cell.

    Domain name: `ibmredbook.com`

13. Configure any additional necessary security items.
14. On the Global security page, enable global security, and save and synchronize your changes.
15. Restart the backup cell.

Security tokens should now flow between the primary and backup cells.

2.8.2 Backup cluster configuration

Figure 2-11 illustrates the configuration of our backup cluster.

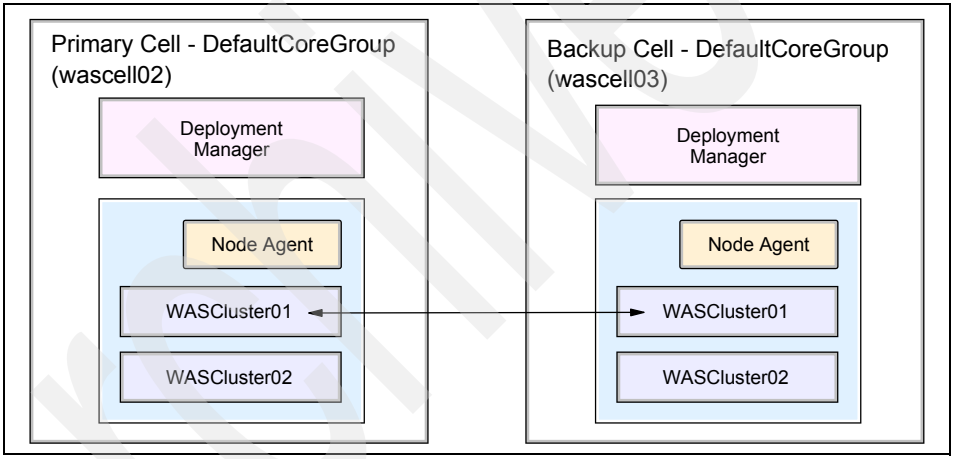


Figure 2-11 Primary and backup cluster

To set up a backup cluster, you need to know the host names and the bootstrap ports of the Deployment Managers for the primary cell and the backup cell. You need to perform configuration on both cells.

1. Go to the primary cell's WebSphere Administrative Console. Click **System administration** → **Deployment manager** → **Ports**. From the list of available ports, check the port number and host name for the BOOTSTRAP_ADDRESS port. Write down these values:
   - Host: `washost01.ibmredbook.com`
   - Port: `9809`

2. Go to the backup cell's WebSphere Administrative Console. Click **System administration** → **Deployment manager** → **Ports**. From the list of available ports, check the port number and host name for the BOOTSTRAP_ADDRESS port. Write down these values:
   - Host: `washost02.ibmredbook.com`
   - Port: `9810`
3. Return to the primary cell's WebSphere Administrative Console. Click **Servers** → **Clusters** → ***Cluster_name*** → **Backup cluster**. Ensure that the name of the backup cluster is the same as the primary cluster.

   Note: When using a split-JVM environment, make sure that you modify the cluster that contains the EJB modules. In our example, this cluster is wascluster01.

4. Click **Domain bootstrap address**. Specify the Host and Port information that you gathered in step 2 (which is the backup cell information). Click **OK a**nd save the changes.
5. Return to the backup cell's WebSphere Administrative Console. Click **Servers** → **Clusters** → ***Cluster_name*** → **Backup cluster**.
6. Click **Domain bootstrap address**. Specify the Host and Port that you looked up in step 1 on page 92 (the primary cell information). Click **OK a**nd save the changes.
7. Restart both clusters.

The EJB clusters in the primary cell and backup cell should point to each other's Deployment Manager in their backup cluster configuration.

Also, refer to the InfoCenter article *Creating backup clusters* available at:

`http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp`

2.8.3 Core group bridge configuration

Note: WebSphere V6.0.2 includes a fix that allows you to set up backup clusters without using the core group bridge support. The configuration is then comparable to previous versions, which means that communication as well as the failover depend solely on an active Deployment Manager, while when using core group bridges, the communication can be spread across all peers.

If you do not want to create core group bridges, you can skip this section because your backup cluster is now set up. Remember that the Deployment Managers must run for the backup cluster failover.

If you want to use the core group bridge service, then configure a core group bridge between the core groups in both cells. This requires access point groups and peer access points between one or more processes in each cell. In our example, we configure bridge points between the Deployment Manager and one Node Agent from each cell as follows:

1. Go to the primary cell's WebSphere Administrative Console. Go to **Servers** → **Core groups** → **Core group bridge settings** → **Access point groups** → **DefaultAccessPointGroup** → **Core group access points**.
2. On the Core group access points page, select **CGAP_1/DefaultCoreGroup**, and click **Show Detail**.
3. On the CGAP_1 page, select **Bridge interfaces**.
4. On the Bridge interfaces page, click **New**.
5. Select the Deployment Manager **wasdmgr02/dmgr/DCS** from under Bridge interfaces. Click **OK.**
6. On the Bridge interfaces page, click **New** to create a second bridge interface.
7. From the Bridge interface menu, select a Node Agent (**wasna01/nodeagent/DCS**). Click **OK.**
8. Save your changes.
9. Switch to the backup cell's WebSphere Administrative Console. Go to **Servers** → **Core groups** → **Core group bridge settings** → **Access point groups** → **DefaultAccessPointGroup** → **Core group access points**.
10. On the Core group access points page, select **CGAP_1/DefaultCoreGroup**, and click **Show Detail**.
11. On the CGAP_1 page, select **Bridge interfaces**.
12. On the Bridge interfaces page, click **New**.
13. From the Bridge interface menu, select the Deployment Manager (**wasdmgr03/dmgr/DCS**). Click **OK**.
14. Click **New** to create a second bridge interface.
15. From the Bridge interface menu, select a Node Agent (**wasna01/nodeagent/DCS**). Click **OK**.
16. Save your changes.

17. Gather the following information about the primary cell:

    a. Find the DCS port for the Deployment Manager on the primary cell. Go to **System administration** → **Deployment manager** → **Ports** → **DCS_UNICAST_ADDRESS**. Write down the port number:

       DCS_UNICAST_ADDRESS: `9352`

    b. Find the DCS port for the wasna01 nodeagent on the primary cell. Go to **System administration** → **Node agents** → **nodeagent** → **Ports** → **DCS_UNICAST_ADDRESS**. Write down the port number:

       DCS_UNICAST_ADDRESS: `9353`

    c. Find the name the of the core group that the EJB cluster belongs to in the cell. Click **Servers** → **Core groups** → **Core group settings**. Select your core group (most likely your core group will be the DefaultCoreGroup), then select **Core group servers**. Verify that your servers are in the selected core group and write down the core group name:

       Core group: `DefaultCoreGroup`

    d. Find the name of the cell. Click **System administration** → **Cell**. Look at the Name field:

       Cell name: `wascell02`

    e. Find the core group access point name. Go to **Servers** → **Core groups** → **Core group bridge settings.** Expand **DefaultAccessPointGroup**. Expand your core group as found in step c, in our case we expand **DefaultCoreGroup**. Write down the value after Core Group Access Point:

       Core Group Access Point: `CGAP_1`

18. Gather the same information for the backup cell:

    a. Find the DCS port for the Deployment Manager on the backup cell. Go to **System administration** → **Deployment manager** → **Ports** → **DCS_UNICAST_ADDRESS**. Write down the port number:

       DCS_UNICAST_ADDRESS: `9353`

    b. Find the DCS port for the wasna01 nodeagent on the backup cell. Go to **System administration** → **Node agents** → **nodeagent** → **Ports** → **DCS_UNICAST_ADDRESS**. Write down the port number:

       DCS_UNICAST_ADDRESS: `9454`

    c. Find the name the of the core group that the EJB cluster belongs to in the cell. Click **Servers** → **Core groups** → **Core group settings**. Select your core group and select **Core group servers**. Verify that your servers are in the selected core group and write down the core group name:

       Core group: `DefaultCoreGroup`

d. Find the name of the cell. Click **System administration** → **Cell**. Look at the Name field:

   Cell name: `wascell03`

e. Find the core group access point name. Go to **Servers** → **Core groups** → **Core group bridge settings.** Expand **DefaultAccessPointGroup**. Expand your core group as found in step c on page 95, in our case we expand **DefaultCoreGroup**. Write down the value after Core Group Access Point:

   Core Group Access Point: `CGAP_1`

19. Return to the primary cell's Administrative Console to create a peer access group to point to the backup cell. On the primary cell, go to **Servers** → **Core groups** → **Core group bridge settings** → **Access point groups** → **DefaultAccessPointGroup** → **Peer access points**. Click **New**.

20. On the Create new peer access point, Step 1 page, you need to enter values into the Name, Cell, Core group and Core group access point fields. Enter any desired name plus the information gathered for the backup cell:

    – Name: `BackupCellGroup`
    – Cell: `wascell03`
    – Core group: `DefaultCoreGroup`
    – Core group access point: `CGAP_1`

    Click **Next**.

21. On the Step 2 page, you need to specify either a peer port or a proxy peer access point. Select **Use peer ports**, then enter the backup cell's Deployment Manager host name and DCS port:

    – Host: `washost02`
    – Port: `9353`

    Click **Next.**

22. On the Step 3 page, confirm the new peer access point, and click **Finish.**

23. Create a second peer port for the Node Agent.

24. On the Peer access points page, select the access point that you just created, **BackupCellGroup/wascell03/DefaultCoreGroup/CGAP_1**, and click **Show Detail.**

25. On the BackupCellGroup page, select **Peer ports** in the Peer addressability box.

26. On the Peer ports page, click **New.**

27. Enter the backup cell's Node Agent host name and DCS port:
    - Host: `washost04`
    - Port: `9454`

    Click **OK**.
28. Save the changes to the primary cell.
29. Switch to the backup cell's Administrative Console to create a peer access group to point to the primary cell. Go to **Servers** → **Core groups** → **Core group bridge settings** → **Access point groups** → **DefaultAccessPointGroup** → **Peer access point**s. Click **New**.
30. On the Create new peer access point, Step 1 page, you need to enter values into the Name, Cell, Core group and Core group access point fields. Enter any desired name plus the information gathered for the primary cell:
    - Name: `PrimaryCellGroup`
    - Cell: `wascell02`
    - Core group: `DefaultCoreGroup`
    - Core group access point: `CGAP_1`

    Click **Next**.
31. On the Step 2 page, you need to specify either a peer port or a proxy peer access point. Select **Use peer ports**, then enter the primary cell's Deployment Manager host name and DCS port:
    - Host: `washost02`
    - Port: `9352`

    Click **Next**.
32. On the Step 3 page, confirm the new peer access point and click **Finish.**
33. Create a second peer port for the Node Agent.

    On the Peer access points page, select the access point just created, **PrimaryCellGroup/wascell02/DefaultCoreGroup/CGAP_1**, and click **Show Detail.**
34. On the PrimaryCellGroup page, select **Peer ports** in the Peer addressability box.
35. Click **New**.
36. Enter the primary cell's Node Agent host name and DCS port:
    - Host: `washost02`
    - Port: `9353`

    Click **OK**.
37. Save the changes to the backup cell.

38. Restart both cells.

Two core group bridges are now configured for the primary and backup cell for the backup clusters to communicate. To review all of the core group bridge settings, go to **Servers** → **Core group bridge settings**. Expand the **DefaultAccessPointGroup** and all items below it to see the group names, host names and ports. Figure 2-12 illustrates this setup.

You can create additional core group bridges by adding them to the DefaultAccessPointGroup and Peer ports on both cells.

You can find additional information in the InfoCenter articles *Configuring the core group bridge between core groups that are in the same cell* and *Configuring the core group bridge between core groups that are in different cells*.

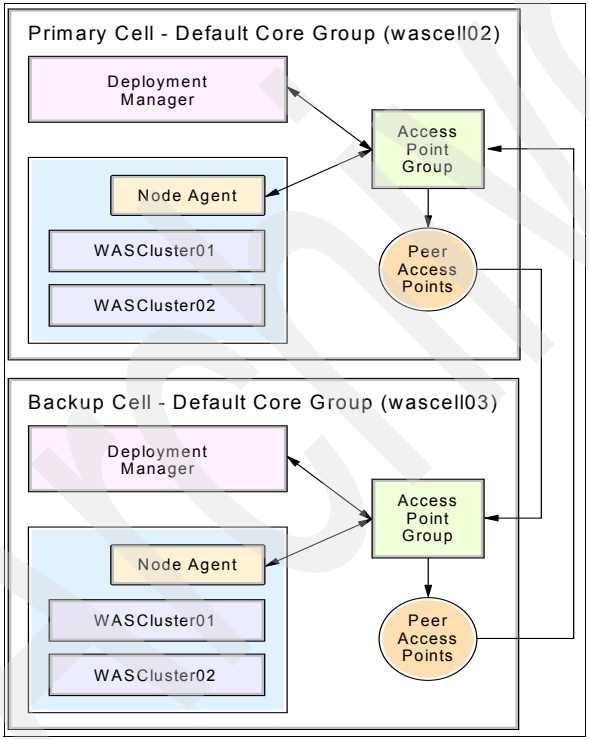


Figure 2-12 Primary and backup clusters with core group bridge

Note: In WebSphere V6.0.2, dynamic core group bridges were introduced. After initially setting up bridges and peer ports, additional bridges can be added without additional corresponding peer ports. The custom property CGB_ENABLE_602_FEATURES is used on the access point group to enable this feature. For more information about this feature see the WebSphere InfoCenter article *Configuring the core group bridge between core groups that are in different cells*.

2.8.4 Testing the backup cluster configuration

Using the Trade 6 application, we split the TradeEJB EJB modules and TradeWeb Web modules across two clusters. After configuring the backup cluster support and core group bridge service, we ran this simple test to verify that the configuration was working:

1. Start all of the Trade clusters in the primary and backup cell.
2. Access the Trade Web site from a server hosting the Web modules in the primary cell:

   `http://wascell02_server_name:9080/trade`
3. Login and buy some stock.
4. Stop the primary Trade EJB cluster from the Administration Console.
5. Open a new browser and access the Trade Web site from a server in the primary cell.
6. Login and buy some stock. This should still succeed. If it fails, review the troubleshooting section.

Additional tests include killing processes that you have assigned to be core group bridges. As long as there is one core group bridge, communication should continue. You can also kill the primary server processes or take down the machines they are running on (as long as the cluster with Web modules stays up) instead of a graceful stop.

To test failback, restart the primary servers. Stop the backup cluster to confirm that work is not being routed to it. If failback does not occur, verify that the backup cluster's information points to the cluster in the primary cell and there is an available core group bridge or the Deployment Manager is up. Review the troubleshooting section.

2.8.5 Troubleshooting

A few troubleshooting tips:

- The easiest part to misconfigure for backup clusters is the core group bridge. There are several settings that must be entered from the other cluster. To review all of the core group bridge settings at once, go to **Servers** → **Core group bridge settings**. Expand the **DefaultAccessPointGroup** and all items below to see the group names, host names, and ports. Compare the primary cell and backup cell to cross check that the access point group from one matches information in the peer access group of the other.
- Review the backup cluster settings for the correct host name and bootstrap address on both the primary and backup cell. Double check that the cluster names are identical.
- Verify that the application is correctly mirrored on the backup cluster. You might need to set up the same resources with the same settings and install the same application (or mirrored resources if the same database will not be used).
- Check that the application works in a non failover mode on both the primary and backup cells.
- Check that the machines in the cells can all contact each other. If they are separated by firewalls, verify that the WebSphere related ports are opened.
- When the primary EJB cluster is down, try accessing the EJBs with a new client or new browser. New connections have the information to connect to the backup cluster.

Reviewing the logs

If failover does not occur, you might see a CORBA.NO_RESPONSE error on the Web or client side as shown in Example 2-14.

Example 2-14 CORBA.NO_RESPONSE error in the log

```
WWLM0061W: An error was encountered sending a request to cluster member
{CELLNAME=wasCell01, CLUSTERNAME=wascluster01,
lsdMemberDistinction=washost1.rchland.ibm.com} and that member has been marked
unusable for future requests to the cluster "", because of exception:
org.omg.CORBA.NO_RESPONSE: Request 60 timed out  vmcid: IBM  minor code: B01
comp.
```

Another error might be the CORBA.COMM_FAILURE shown in Example 2-15:

Example 2-15 CORBA.COMM_FAILURE error in the log

```
WWLM0061W: An error was encountered sending a request to cluster member
{MEMBERNAME=wasmember01, NODENAME=wasnode01} and that member has been marked
unusable for future requests to the cluster "", because of exception:
org.omg.CORBA.COMM_FAILURE: CONNECT_FAILURE_ON_SSL_CLIENT_SOCKET - JSSL0130E:
java.io.IOException: Signals that an I/O exception of some sort has occurred.
Reason: Connection refused vmcid: 0x49421000 minor code: 70 completed: No
```

If you receive these errors and you enabled core group bridges, then verify that you receive messages of processes joining the access point groups as shown in Example 2-16. If you do not see these messages, review the core group bridge settings.

Example 2-16 Process joining messages for access point groups

```
DCSV1032I: DCS Stack DefaultAccessPointGroup at Member 10.0.4.2:9353: Connected
a defined member 10.0.4.1:9352
```

Security

If you have security related problems, verify or review the following:

1. Security related messages in the application server logs.
2. You have the same user registry on both cells.
3. LTPA tokens were exported from the primary cell and imported in the backup cell.
4. After enabling security or making security configuration changes, the cell was synchronized and restarted.
5. The time matches on all machines in both cells.
6. Security works if the application is installed on a single cluster.

3

WebSphere administrative process failures

This chapter explains what the effects are for the WebSphere environment if an administrative process, such as the Deployment Manager or a Node Agent fails (this also includes planned downtime of one of these processes). The Deployment Manager and Node Agents cannot automatically failover to other systems in the WebSphere environment and are thus considered single points of failure.

However, after reading this chapter, you might find that failures of these administrative processes are not disastrous for your environment and you can live with an outage for a certain amount of time.

In case you cannot afford a failure, you can make these processes highly available using clustering software, such as IBM HACMP, IBM Tivoli System Automation, VERITAS Cluster Server, or Sun Cluster. Setting up clustered environments is covered in great detail in Part 5, “Using external clustering software” on page 283.

Failures of application servers are not covered in this chapter. See Chapter 2, “WebSphere Application Server failover and recovery” on page 35 for information about this topic.

3.1 Introduction to process failures

The Node Agents and the Deployment Manager are administrative processes in the WebSphere environment. One of their main responsibilities is to keep configuration data synchronized. Node Agents and the Deployment Manager both use a repository of XML files on their own nodes. The master repository data is stored on the Deployment Manager node. That data is then replicated to each node in the cell. The default synchronization interval is 60 seconds. The synchronization process is unidirectional from the Deployment Manager to the Node Agents to ensure repository integrity. That means that any changes made on the Node Agent level are only temporary and will be overwritten by the Deployment Manager during the next synchronization. Only changes to the master data through the Deployment Manager are permanent and replicated to each node.

Other tasks for the Deployment Manager are related to the Tivoli Performance Viewer, backup clusters, JMX routing, distributed logging, the naming server and the security server. To learn what impact a failure of the Deployment Manager has on the various WebSphere system components, see 3.2, “Deployment Manager failures” on page 104.

In addition to the file synchronization services, the Node Agent also provides runtime support for the Location Service Daemon (LSD), JMX server, distributed logging, naming server, security server, and is required for starting application servers. See 3.3, “Node Agent failures” on page 111 for details.

You learn in this chapter that, depending on your requirements and WebSphere environment, the impact of a failure of one of these processes might be minimal. Some customers even run their production environment without an active Deployment Manager.

3.2 Deployment Manager failures

The Deployment Manager is not clustered and therefore a single point of failure. However, its impact on the application client processing is limited because all configuration data is replicated to every Node Agent in the cell. As soon as the Deployment Manager server becomes available again, all Node Agents in the cell will discover it, and all functions will return to normal.

Also, in WebSphere V6, the role of the Deployment Manager during runtime operation of a WebSphere system is less critical than in previous versions. The Deployment Manager is no longer responsible for keeping routing tables up to date for clustered resources such as J2EE applications or messaging engines. The routing table logic now runs on an elected member of the cluster itself. If that

member fails, then the HAManager moves the task to a surviving cluster member. The routing table is thus now fault tolerant. However, the Deployment Manager is required for the following tasks:

- Making configuration changes (permanent or runtime) and synchronizing these with the cell.

  The Deployment Manager is the central control point for all administrative and configurational tasks as well as operational actions. It hosts the Administrative Console application which is used to configure the entire cell.

- JMX routing through the Deployment Manager to manage application servers and clusters, to deploy applications to a cluster, and so forth.

  Some of these configuration tasks can be done using **wsadmin** commands directly on the Node Agent or application server. However these changes are not permanent and are overwritten based on the master repository configuration when the Deployment Manager becomes available again. See 3.2.9, “Administrative clients” on page 109 for more information. For details on JMX and distributed administration see Chapter 3 of *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

- Performance Monitoring

  The Tivoli Performance Viewer is integrated into the Administrative Console in WebSphere V6.

- Backup cluster failover. See 2.7, “Backup cluster support” on page 87.

Failure causes

What can cause the Deployment Manager to become unavailable? The following are failure causes:

- Expected server process failure, for example stopping the server.
- Unexpected server process failure, for example the server JVM crashes, simulated by **kill -9**.
- Server network problem, for example a network cable is disconnected or a router is broken.
- Unexpected and expected machine problem, for example a machine shutdown, operating system crashes, or power failures.
- Disk failure and thus the configuration cannot be read.

The following sections discuss the behaviors of Deployment Manager server failures and how WebSphere V6 mitigates the impacts of these failures on the WebSphere environment.

3.2.1 Configuration management

In WebSphere V5 and higher, each individual node is "added" to the Deployment Manager cell. The Deployment Manager holds the cell's master configuration repository in the form of XML files. A subset of the master repository (related to a node's configuration) is then replicated to all the nodes in the cell. The Deployment Manager keeps all copies of the configuration files synchronized across all nodes. During normal operation, the Deployment Manager consistently synchronizes the repository data on each node with the master repository. The Deployment Manager also synchronizes application binaries to all application servers in a cluster (through the Node Agent).

You can turn synchronization on or off and change configuration parameters to tune synchronization using the Administrative Console (**System Administration** → **Node Agents** → ***Node_Agent_Name*** → **File synchronization service**).

WebSphere V6 also provides the option to synchronize copies of the repository and application binaries manually by using the **syncNode** command:

```
syncNode deploymgr_host deploymgr_port [options]
```

You can also use the Administrative Console (**System administration** → **Nodes**) as shown in Figure 3-1.

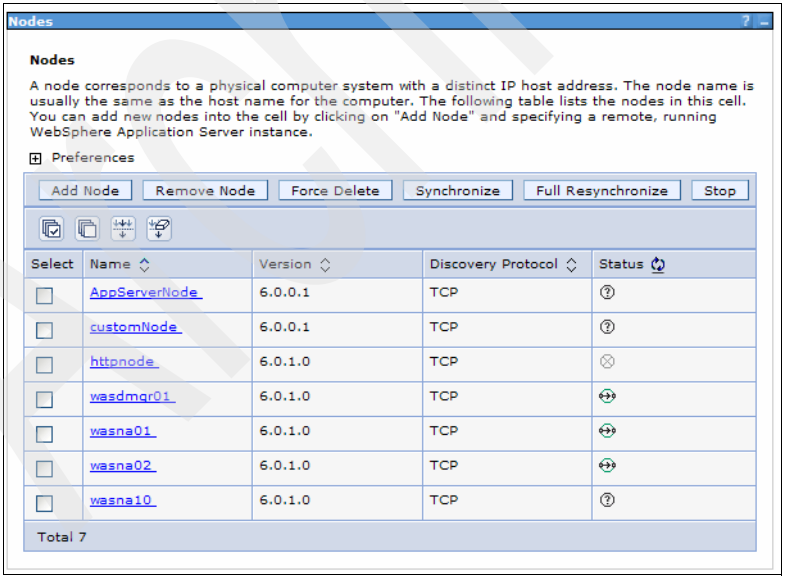


Figure 3-1 Node synchronization in the Administrative Console

Note: If the Deployment Manager is not available, the repository cannot be synchronized. However, you can use **wsadmin** commands to change the configuration at the node level directly. Be aware that repository files across all nodes in the domain might become inconsistent or stale when doing so. When the Deployment Manager comes online again, all changes made locally to Node Agents or application servers are lost and are overwritten with the master repository.

3.2.2 Node Agent

The Node Agent interacts directly with the Deployment Manager to perform all administrative and configuration tasks. If the Deployment Manager is unavailable, all cell-wide tasks cannot be executed and all tasks done locally will be overwritten by the master repository when the Deployment Manager becomes available again.

When the Deployment Manager is not available, configuration synchronization with the Node Agent fails with the following error message:

```
[6/30/05 9:25:56:061 CDT] 0000002d NodeSyncTask E   ADMS0015E: The
synchronization request cannot be completed because the node agent cannot
communicate with the deployment manager.
```

The Deployment Manager and the Node Agents can be started in any order. They will discover each other as soon as they are started.

3.2.3 Application server

Application servers have no direct interaction with the Deployment Manager. Thus, application servers can serve client requests regardless of Deployment Manager availability. However, they rely on up-to-date configuration data and on the fact that the Deployment Manager server replicates and initiates any changes.

3.2.4 Naming server

The Deployment Manager also hosts a cell naming server that controls the whole naming context structure.

Clients can bootstrap to the cell naming server using:

```
prop.put(Context.PROVIDER_URL, "corbaloc::dmgrhost:9809");
```

The Deployment Manager server is not workload managed in WebSphere V6. In addition, the naming service is not inside the Deployment Manager process.

Unless you have a highly available Deployment Manager using clustering software such as IBM HACMP or TSA, you should not use it to bootstrap your application clients. A highly available Deployment Manager has the advantage of a single image of the domain.

No matter where your client bootstraps, any naming binding or update writing at cell scope fails if the Deployment Manager is not available.

3.2.5 Security service

The cell security server is not available when the Deployment Manager is down. This has no impact on application clients because they use the local security service in every application server. Nevertheless, the LDAP server does present a single point of failure for security servers. See 15.6, “LDAP Server” on page 580 for more information about high availability of LDAP servers.

3.2.6 Application clients

There is very limited direct impact on application clients when the Deployment Manager is not available. They will notice the failure only if they are bootstrapped to the cell naming server or when the application clients do bindings and updates. However, application clients might be impacted indirectly by unavailable configuration and operational services due to the failed Deployment Manager, for example inconsistent configuration data or aged application binaries.

3.2.7 Synchronization Service and File Transfer Service

The Deployment Manager provides file transfer and synchronization of configuration files and application binaries across nodes in the domain. These services are not available when the Deployment Manager is down.

3.2.8 RAS Service and PMI monitoring

The cell-wide information and remote logs are not available when the Deployment Manager is unavailable.

3.2.9 Administrative clients

This section discusses the impacts of Deployment Manager failures on the Administrative Console and the **wsadmin** scripting interface.

Administrative Console

The default **adminconsole** application cannot be started if the Deployment Manager is unavailable.

The wsadmin scripting interface

By default, **wsadmin** connects to the Deployment Manager server. If it is unavailable, **wsadmin** fails. You can change the wsadmin.properties file or add parameters on the command line to attach **wsadmin** to a Node Agent or application server.

You can look up the **wsadmin** connect types and ports in the SystemOut.log file, for example, for an application server:

```
The SOAP connector is available at port 8881
The RMI Connector is available at port 9811
```

For example, you can use the **wsadmin** interface with the following parameters:

```
wsadmin -conntype RMI -host localhost -port 9811
```

Alternatively, you can modify the wsadmin.properties file with the application servers' or Node Agents' host and port as shown in Example 3-1. The wsadmin.properties file is located in the following directory:

<*install_root*>/profiles/*profilename*/properties

Example 3-1 The wsadmin.properties file

```
...
# The connectionType determines what connector is used.
# It can be SOAP or RMI.
# The default is SOAP.
#-------------------------------------------------------------------------
#com.ibm.ws.scripting.connectionType=SOAP
#com.ibm.ws.scripting.connectionType=null
com.ibm.ws.scripting.connectionType=soap
#com.ibm.ws.scripting.connectionType=RMI
#-------------------------------------------------------------------------
# The port property determines what port is used when attempting
# a connection.
# The default SOAP port for a dmgr or custom profile is 8879
#-------------------------------------------------------------------------
#com.ibm.ws.scripting.port=8879
#com.ibm.ws.scripting.port=null
```

```
com.ibm.ws.scripting.port=8881
#--------------------------------------------------------------------------
# The host property determines what host is used when attempting
# a connection.
# The default value is localhost.
#--------------------------------------------------------------------------
#com.ibm.ws.scripting.host=localhost
#com.ibm.ws.scripting.host=mySystem
#com.ibm.ws.scripting.host=localhost
com.ibm.ws.scripting.host=mySystem.ibmredbook.com
...
```

Remember that any change made to the Node Agent or the application server is lost when the Node Agent synchronizes its repository with the master repository when the Deployment Manager comes up again.

3.2.10 Enhancing Deployment Manager availability

A failure of the Deployment Manager is not critical for the functioning of a WebSphere environment. However, there are several options to enhance the availability of the Deployment Manager, such as making it a monitored process or using clustering software. See 3.4, “Restarting WebSphere processes as an OS service” on page 117 and 3.5, “Enhancing WebSphere process availability using clustering software” on page 118 for information.

Another option is the approach that is described in the online article *Implementing a Highly Available Infrastructure for WebSphere Application Server Network Deployment, Version 5.0 without Clustering* by Tom Alcott, which is available at:

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

3.3 Node Agent failures

The Node Agent is an administrative process and is not involved in application serving functions. It hosts important administrative functions such as file transfer services, configuration synchronization and performance monitoring. The Node Agent is required to be started before any application servers on that node can be started and contains the Location Service Daemon (LSD) in which each application server registers on startup.

The following types of errors in the process, network, disk, and machine might cause a Node Agent server to fail:

- Expected server process failure, for example stopping the server.
- Unexpected server process failure, for example the server JVM crashes, simulated by **`kill -9`**.
- Server network problem, for example the network cable is disconnected or a router is broken.
- Unexpected and expected machine problems, for example a machine shutdown, operating system crashes, or power failures.
- Disk failure and thus the configuration cannot be read.

The following sections discuss the behaviors of Node Agent failures and how WebSphere V6 mitigates the impacts of these failures on application servers and application clients.

3.3.1 Application servers

The Node Agent hosts the Location Service Daemon, publishes and synchronizes configuration data to other application servers, and monitors and launches the managed application server processes.

Starting application servers

An application server can only be started if the local Node Agent is available. If the Node Agent is not active, the application server cannot register itself with the LSD, and you receive the error message shown in Example 3-2.

Example 3-2 Error message at application server startup with inactive Node Agent

```
...
[6/30/05 9:03:29:208 CDT] 0000000a WsServerImpl  E   WSVR0009E: Error occurred
during startup META-INF/ws-server-components.xml
[6/30/05 9:03:29:218 CDT] 0000000a WsServerImpl  E   WSVR0009E: Error occurred
during startup com.ibm.ws.exception.RuntimeError:
com.ibm.ws.exception.RuntimeError: com.ibm.ejs.EJSException: Could not register
```

```
with Location Service Daemon, which could only reside in the NodeAgent. Make
sure the NodeAgent for this node is up an running.; nested exception is:
        org.omg.CORBA.ORBPackage.InvalidName:
LocationService:org.omg.CORBA.TRANSIENT: java.net.ConnectException: Connection
refused:host=system1.ibmredbook.com,port=9100  vmcid: IBM  minor code: E02
completed: No
...
```

For active application servers

When the application server is started, it runs normally even if the Node Agent becomes unavailable. However, configuration data cannot be updated.

Restarting application servers automatically

Application server processes are managed by the Node Agent. When the Node Agent is unavailable, the application server loses the benefit of being a managed server. One of these benefits is that an application server is restarted automatically if it dies unintentionally.

You can tune the application server process monitoring parameters on the Administrative Console by changing the Monitoring Policy. To do this, go to **Servers** → **Application servers** → ***AppServer_Name*** → **Java and Process Management** → **Monitoring Policy.** Figure 3-2 shows the configuration panel.

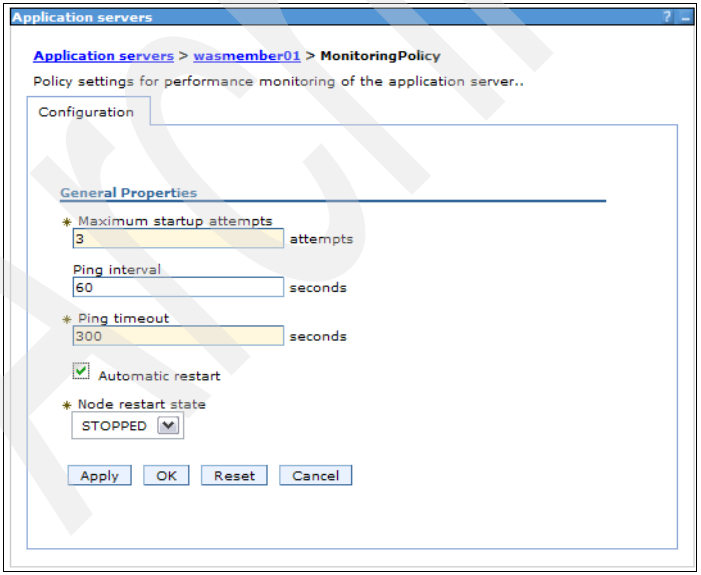


Figure 3-2 Managed server monitoring parameters tuning

Network failures: loopback alias configuration

The Node Agent normally pings all application servers on the node and restarts application servers that are not available. If the system is disconnected from the network, for example due to a network adapter or cable problem, this communication between the Node Agent and its application servers depends on the loopback port.

Basically, when the system is disconnected from the network, the Node Agent cannot ping the application server unless the loopback port address (127.0.0.1) is configured as an alias endpoint for the machine. The reason for this is that the loopback port is only considered a valid port for the application server when the loopback port address is configured as an alias for the system.

If the Node Agent cannot ping the application server for a certain amount of time, then it tries to restart the application server based on the settings in the Monitoring Policy.

The following scenario explains why it is important to configure the loopback alias. Assume that there is a network failure and the ping from the Node Agent to the application server fails. The Node Agent thinks that the application server is down, even though it is not.

- If the network is restored *prior* to the Node Agent, it thinks that the application server has failed, then everything is fine.
- If the network is restored *after* the Node Agent, it thinks that the application server is down. Then, the Node Agent tries to restart the application server, meaning it would try to start a second copy of the application server, which of course fails. If the network is restored after the Node Agent gives up trying, then the Node Agent thinks that the application server is down, even if it is up.

To configure the loopback alias, add the following line to the hosts file:

```
127.0.0.1 hostname
```

3.3.2 Deployment Manager

Node Agents and the Deployment Manager work together to manage administrative tasks and to make configuration and operational changes. If a Node Agent is not available, the Deployment Manager cannot inform the Node Agent of these changes. However, when the Node Agent becomes available again, all missing information is sent to the Node Agent during the first synchronization.

3.3.3 Security service

It is very important that all Node Agents in a cell are up and running when you change security related settings, for example, when you enable global security or change user registry settings. Otherwise, synchronization with the Deployment Manager might fail due to inconsistent or missing security information when the Node Agent comes back up.

Other than this, a failure of a Node Agent does not have any impacts on an already set up and running secured environment because each application server JVM hosts a security service. The security service uses the security settings held in the configuration repository to provide authentication and authorization functionality.

3.3.4 Naming server

In IBM WebSphere Application Server Network Deployment V6, a naming server exists in every server process (application server, Node Agent, and Deployment Manager). You can bootstrap your client to any of these naming servers. Usually servlets, EJB clients, and J2EE clients start their bootstrap from their local application server and end up on the server where the objects are found.

For Java clients, you can bootstrap to more than one naming server on different Node Agents, as shown in Example 3-3.

Example 3-3 Java client bootstrapping

```
prop.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");
prop.put(Context.PROVIDER_URL, "corbaloc::host1:2809,:host2:2809");
Context initialContext = new InitialContext(prop);
try { java.lang.Object myHome =
initialContext.lookup("cell/clusters/MyCluster/MyEJB");
myHome = (myEJBHome) javax.rmi.PortableRemoteObject.narrow(myHome,
myEJBHome.class);
} catch (NamingException e) { }
```

The client tries the bootstrap servers on host1 and host2 automatically until it gets an InitialContext object. The order in which the bootstrap hosts are tried is not guaranteed.

The Node Agent is not workload managed. If a client gets a cached InitialContext object and that Node Agent is unavailable, the InitialContext object will not automatically failover to another Node Agent, so the lookup with this InitialContext object fails. In order to avoid this problem, you need to disable the

cache in your client by supplying the following property when initializing the naming context:

```
prop.put(PROPS.JNDI_CACHE_OBJECT, PROPS.JNDI_CACHE_OBJECT_NONE)
```

Where `PROPS.JNDI_CACHE_OBJECT` is a Java constant defined in com.ibm.websphere.naming.PROPS.

Or more conveniently by setting the Java command line property as:

```
java -Dcom.ibm.websphere.naming.jndicacheobject=none MyClient
```

Using Node Agents as bootstrap servers is not recommended because they are not workload managed. Because application servers in a cluster are workload managed, you should use application servers as bootstrap servers instead as follows:

```
prop.put(Context.PROVIDER_URL, "corbaloc::host1:9810,:host1:9811,
:host2:9810, :host2:9811");
```

This discussion is only relevant for naming lookup (read) operations. If your application is binding (writing), a failure occurs if the Node Agent is unavailable, because the Node Agent coordinates the binding process.

Refer to *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392 for more information about bootstrapping.

3.3.5 Application clients

An application client receives its routing information from the LSD hosted in the Node Agent. If the application server environment does not have routing information for a client failover available, the client will fail to run if all Node Agents are unavailable. When the application server environment has routing information available, the application client will run successfully if it is not using a single Node Agent to bootstrap and is not binding (writing). All EJB IORs contain the list of LSD host and ports.

3.3.6 Synchronization service and File transfer service

These services fail to function when the Node Agent is unavailable.

3.3.7 RAS service, PMI and monitoring

A node with a failed Node Agent is not able to provide RAS service, PMI, and monitoring information to the cell.

3.3.8 Administrative clients

There are two kinds of administrative clients: the Administrative Console and the **wsadmin** scripting interface. The **wsadmin** tool scripting interface can be attached to any server (Node Agent, Deployment Manager, or application servers). The Administrative Console is a Web application that is installed in the Deployment Manager in a WebSphere Network Deployment environment.

Administrative Console

The following limitations when working with the Administrative Console apply when a Node Agent is not available:

- Any query for any information about that node fails and leaves the server status for that node as *unknown* or *unavailable*.
- Any configuration changes do not appear on that node until the Node Agent is restarted.
- Any operational actions (start, stop, delete, create) from the Administrative Console to the servers on the node with a failed Node Agent fails.

However, you can do any administrative and configuration tasks using the Administrative Console for other nodes where the Node Agents are still alive. Configuration changes are synchronized with the Node Agent when it becomes available again.

The wsadmin scripting interface

You can connect the **wsadmin** scripting interface to every server in the WebSphere cell. By default, **wsadmin** is attached to the Deployment Manager. You can change the default by editing the wsadmin.properties file located in the <*install_root*>/profiles/*profilename*/properties directory or you can specify the conntype, host, and port parameters in the command line:

```
wsadmin -conntype SOAP -host mynode -port 8878
```

You can look up the conntype and port in the SystemOut.log. For example:

```
JMXSoapAdapte A ADMC0013I: The SOAP connector is available at port 8878
RMIConnectorC A ADMC0026I: The RMI Connector is available at port 2809
```

Naturally, **wsadmin** cannot connect to the Node Agent when it is unavailable.

3.3.9 Enhancing Node Agent availability

In WebSphere Network Deployment, we usually have more than one Node Agent in the cell, and all Node Agents host the Location Service Daemon (LSD). So in case of a Node Agent failure, another Node Agent can service client

requests. Therefore, having more than one Node Agent in a cell is the basic solution to ensure LSD availability if a Node Agent process is lost.

Other availability improvement options

There are several other options to enhance the availability of Node Agents, such as making them a monitored process or using clustering software. See 3.4, “Restarting WebSphere processes as an OS service” on page 117 and 3.5, “Enhancing WebSphere process availability using clustering software” on page 118 for information.

3.4 Restarting WebSphere processes as an OS service

As a first step in making HA provisions for the Deployment Manager or Node Agent, you should make use of the platform specific mechanisms for providing operating system process monitoring and restart. When you have added the Deployment Manager and Node Agent processes as operating system monitored processes, the operating system restarts them automatically in case of an abnormal termination. This minimizes the impact of an outage due to a process failure. For further details, refer to the InfoCenter article *Automatically restarting server processes* at:

```
http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp
```

Windows

In Windows, you can use the **WASService** command to create a Windows service for any WebSphere Application Server Java process. Adding the service is also possible during installation of the product but you can always use the **WASService** command to add additional processes later or in case you did not add the Windows service during installation.

Refer to the InfoCenter article *WASService command* for detailed instructions on how to use the **WASService** command.

UNIX and Linux - Deployment Manager

The steps to achieve the same in UNIX® or LINUX are as follows:

1. Navigate to the bin directory under the Deployment Manager’s profile:

```
cd /IBM/WebSphere/AppServer/profiles/dmgr_profile_name/bin
```

2. Run the **startManager.sh** command with the **-script** option, as shown in Example 3-3 on page 114

Example 3-4 startManager.sh -script command

```
./startManager.sh -script
ADMU0116I: Tool information is being logged in file
/IBM/WebSphere/AppServer/profiles/itsoprofile01/logs/dmgr/startServer.log
ADMU0128I: Starting tool with the itsoprofile01 profile
ADMU3100I: Reading configuration for server: dmgr
ADMU3300I: Launch script for server created: start_dmgr.sh
```

3. Edit the rc.was file under *<install_root>*/bin and set the value of the launchScript variable to the script created in step 2.

```
launchScript=start_dmgr.sh
```

4. Add an entry to the /etc/inittab file:

```
dm:23:once:/IBM/WebSphere/AppServer/bin/rc.was >dev/console 2>&1
```

Note: You must have root authority to edit the inittab file.

UNIX and Linux - Node Agent

Follow the instructions from "UNIX and Linux - Deployment Manager" on page 117 but navigate to the Node Agents' profile /bin directory and use the **startNode.sh** command instead of the **startManager.sh** command.

3.5 Enhancing WebSphere process availability using clustering software

Operating system process monitoring and restart does not help if the Deployment Manager or Node Agent fails due to any network, disk, operating system, or host machine problem. In order to achieve 24 x 7 availability, third-party clustering software such as IBM HACMP, IBM Tivoli System Automation, Sun Cluster, or VERITAS Cluster Server needs to be used.

The underlying concept for all of the aforementioned third-party software is essentially the same and is described in Chapter 9, "Configuring WebSphere Application Server for external clustering software" on page 285. Then, we provide individual chapters for each of these software packages:

- Detailed configuration instructions for using IBM HACMP to create a highly available Deployment Manager or Node Agent and application server cluster are provided in Chapter 11, "WebSphere and IBM HACMP" on page 417.

- Detailed configuration instructions for using IBM Tivoli System Automation (TSA) to create a highly available Deployment Manager or Node Agent and application server cluster are provided in Chapter 10, “WebSphere and IBM Tivoli System Automation” on page 367.
- Detailed configuration instructions for using VERITAS Cluster Server to create a highly available Deployment Manager or Node Agent and application server cluster are provided in Chapter 12, “WebSphere and VERITAS Cluster Server” on page 445.
- Detailed configuration instructions for using Sun Cluster to create a highly available Deployment Manager or Node Agent and application server cluster are provided in Chapter 13, “WebSphere and Sun Cluster” on page 483.

4

High availability system administration

This chapter describes system administration in a highly available WebSphere Application Server environment. It helps you to understand the administration tasks that are required during an hardware upgrade, hardware replacement, and WebSphere software version upgrade while still maintaining a highly available system. We describe the procedures for administering these three scenarios via the Administrative Console and also provide a list of **wsadmin** scripts to automate the administration tasks.

Administration of HTTP requests and Web servers are external to WebSphere Application Server and are not covered in this chapter. We do discuss, however, what you should consider regarding the Web server plug-in communication to the application servers when stopping and restarting application servers.

4.1 Introduction to high availability

A high availability (HA) WebSphere system is made up of two or more machines or LPARs that host application servers. Each machine or LPAR is considered a node in the WebSphere environment. Each node hosts one or more application servers (cluster members) which are interconnected in a cluster that hosts the same application. Failover and workload distribution is provided among the members of the cluster.

IBM WebSphere Application Server Network Deployment V6 provides a high availability framework that eliminates single points of failure for most WebSphere processes. The HAManager together with the WebSphere workload management (WLM) component provide redirection of requests within the cluster when a failure of a node or process is detected.

Refer to Chapter 2, “WebSphere Application Server failover and recovery” on page 35 and *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392 for more information about failover and WebSphere WLM. Refer to Chapter 6, “WebSphere HAManager” on page 175 for information about the HAManager.

As an HA administrator, you should be aware of the different WebSphere components that participate in the HA environment and the impact when they are not available due to a hardware or software upgrade. Planning ahead for the right hardware capacity to provide high availability support and a carefully derived schedule for software upgrades are important to avoiding the risk of a failure without failover coverage.

It is imperative that the remaining nodes are capable of handling all the workload when one of them is unavailable. This way, you avoid overloading the running servers and minimize unavailability. So, a highly available environment always needs some overcapacity.

Note: Perform a software upgrade when there is minimal traffic to the system and automate the process in a production environment.

4.1.1 System setup for the administration scenarios

Figure 4-1 on page 123 shows the full cell configuration of our environment, with multiple nodes, clusters, and cluster members. For simplicity reasons, we focus on two nodes, wasna01 and wasna02, for the scenarios that this chapter describes. Each node is configured to be part of two clusters (wascluster01 and

wascluster02), and each cluster has multiple cluster members (wasmember01 through 06). Figure 4-2 on page 124 illustrates this subset of the entire cell.

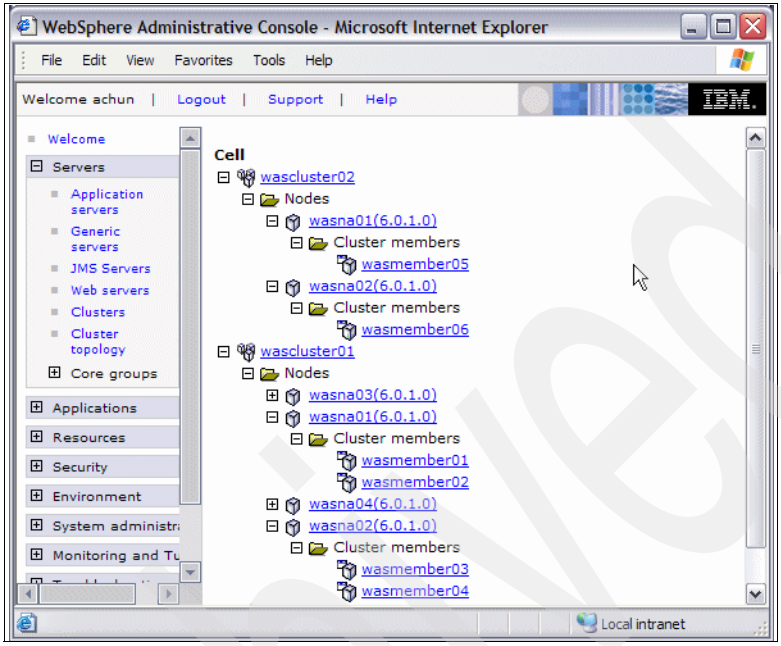


Figure 4-1 Cell configuration of the redbook environment

Figure 4-2 on page 124 shows the test environment using only the two nodes that we used during our tests. This setup illustrates the various adminstration tasks that are involved in the following administration scenarios:

- Hardware replacement
- Hardware upgrade
- WebSphere version upgrade (installing refresh packs)

We explain these scenarios mainly using the Administrative Console.

In addition, you can find a list of **`wsadmin`** scripts that are downloadable from the IBM developerWorks® Web site in 4.6, “Sample wsadmin scripts for administration tasks” on page 139. You can use these scripts to automate some of the common administration tasks in a WebSphere environment.

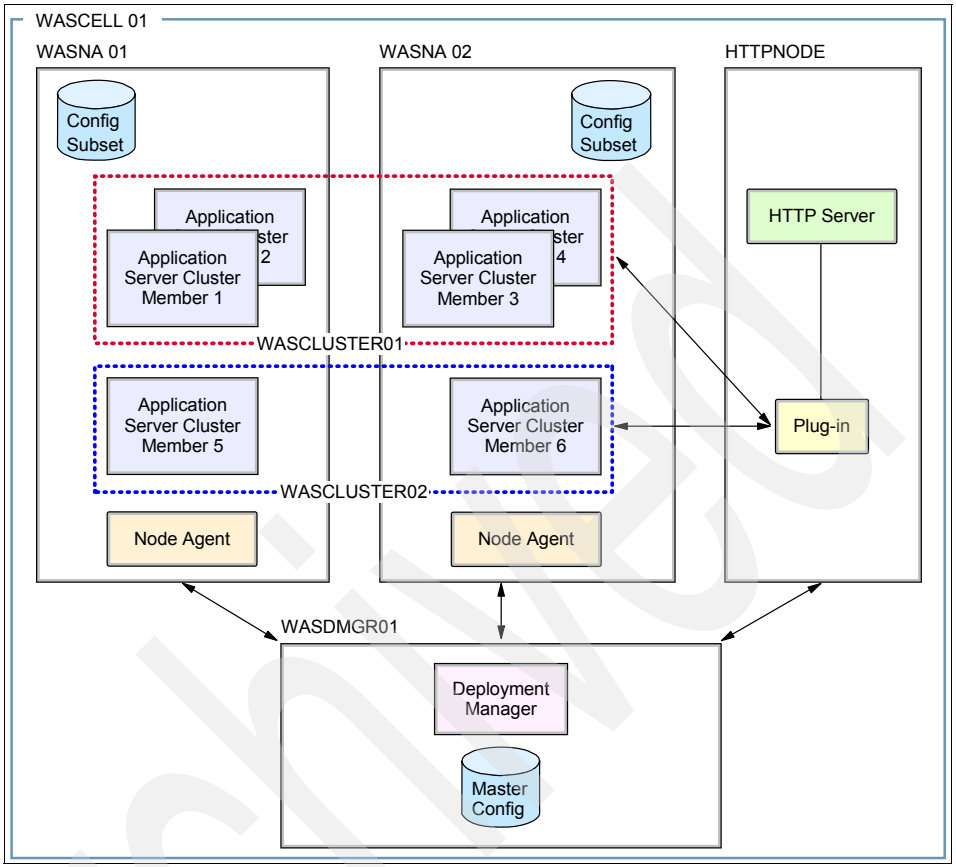


Figure 4-2 HA system and configuration setup

WebSphere Application Server V6 allows for integration of WebSphere into an environment that uses other high availability frameworks, such as IBM HACMP, Tivoli System Automation, VERITAS Cluster Server, or Sun Cluster to manage a highly complex HA system. In this chapter, we focus on the HA setup and configuration available within WebSphere itself. For information about the external HA options, refer to Part 5, “Using external clustering software” on page 283.

4.2 Starting or stopping application servers and the Web server plug-in retry interval

When stopping and restarting application servers, it is important to consider the correlation with the Web server plug-in retry interval. This is especially important in an environment with only a few application servers.

You need to understand the rate between starting and stopping application servers in a cluster and the retry interval so that you do not get HTTP request failures when you stop application servers in a cluster. The retry interval tells the Web server plug-in how long to wait before retrying an HTTP request against a specific application server. If you stop and start application servers in a cluster faster than the retry rate, then a situation can occur where the plug-in assumes that two or more application server are down, when in fact they are not.

Here is an example of this situation. Let us assume the following:

- The retry interval is set to 60 seconds (which is the default).
- There are two application servers in the cluster.
- We are running a servlet application.
- We alternate taking the two servers down. We take them down and restart them in a 45 second interval.

The Web server plug-in sends a request to a specific application server and if that server is unavailable, it marks it as down. The request is then sent to the next application server in the cluster. The plug-in does not retry the marked down server until the retry interval has passed.

In our example, because the retry interval is longer than the recycle time, there are time slots where one server's retry count has not yet expired. So, the plug-in does not retry that server (even though it might be back up), and the other server is down. Thus, there might be a time slot where the Web server plug-in thinks that both servers are down. Figure 4-3 on page 126 illustrates this example.

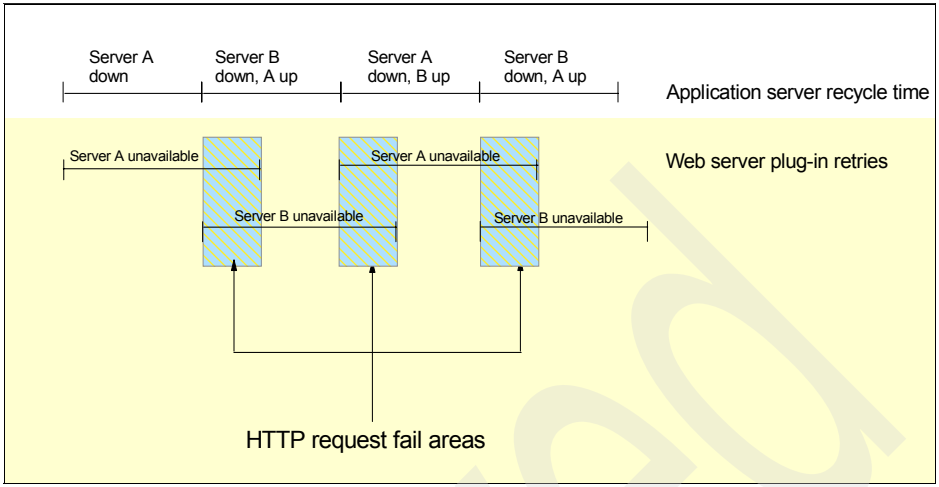


Figure 4-3 Correlation between retry interval and stopping servers

In an environment with two application servers per cluster, the plug-in cannot forward the request to any application server and, thus, returns an HTTP request failure. If there are more than two application servers in the cluster, the plug-in routes the request to the remaining active servers in the cluster. However, depending on the difference between the retry interval and the application server recycle time, there might be more than two application servers that are perceived to be unavailable at any given time.

To avoid this problem, increase the recycle rate to be at least as long in duration as the retry interval. If your retry interval is 60 seconds, then delay at least 60 seconds between starting one application server and stopping another application server. You can verify the retry interval setting either by using the Administrative Console (**Servers** → **Web servers** → ***WebServer_Name*** → **Plug-in properties** → **Request Routing)** or by looking at the plugin_cfg.xml file.

If it is not possible to add an appropriate delay between recycling servers, then ensure that your remaining application server environment has the capacity to handle requests for the application assuming that two (or more) of the application servers are unavailable at the same time.

So, there are two maxims to remember:

1. Do not stop an application server in a shorter duration than the retry interval.
2. If you stop an application server in a shorter duration than the retry interval, ensure that the set of remaining active servers can handle the capacity of having two application servers unavailable for a period of time equal to the retry interval.

4.3 Replacing hardware

In this scenario, we are replacing the hardware which hosts the wasna01 node.

Note: This scenario covers planned maintenance. Thus, you should not use this in case of a hardware problem. If you need to recover from a hardware failure by replacing the hardware, you can follow the description from Chapter 14, “Backup and recovery of Network Deployment configuration” on page 549. The approach described there requires a recent backup of your WebSphere configuration using **`backupConfig`** or a file system backup.

The new hardware is configured to contain the exact same configuration as the removed node when it is defined in the cell. Obviously, to ensure HA, the node to be replaced must not be the only physical machine in the environment and the remaining systems should be able to handle the additional workload.

We recommend reading the entire section before performing the task.

4.3.1 Removing the node from the cell

In order to replace the hardware, you should first ensure that all processes on the node are stopped:

1. Select **Servers** → **Application servers** and select all application servers that belong to node wasna01, as shown in Figure 4-4 on page 128. Then, click **Stop**. In our example the servers to stop are wasmember01 and wasmember02 (which are part of wascluster01) as well as wasmember05 (which is part of wascluster02).

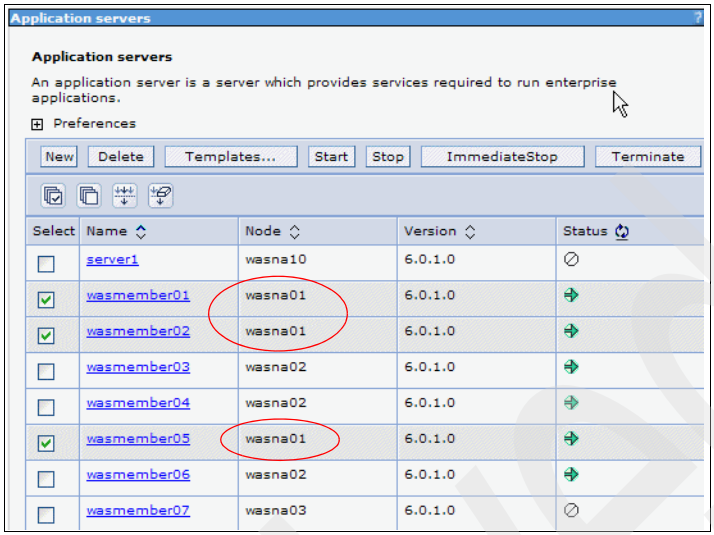


Figure 4-4 Select the application servers that belong to node wasna01

Depending on the number of application servers that you have chosen, it might take several minutes to stop all application servers. The Server status feedback panel gives you continuous information about the stop procedure. When all processes have stopped, click **OK** to proceed.

Attention: When using an application that uses the default messaging provider, such as Trade 6, it is important to drop all tables that were created for the messaging engines (MEs). The MEs have a UUID assigned to them upon creation/instantiation and when creating a new application server (cluster member), there is a UUID mismatch, and the application server cannot assign the ME.

You can obtain the schema names used for each of your messaging engines by selecting **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. Connect to the applications' database and list the tables for each of the schemas. Then, drop all tables listed for each schema.

For more information, visit 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`

2. You can now remove the node from the cell configuration. Click **System administration** → **Nodes**. Select **wasna01** and click **Remove Node**. Note that the Node Agent must be active for this procedure to work.

   Click **OK** on the Remove node confirmation screen. Verify that there is no error on the console. **Save** your changes.

   Tip: Alternatively, you can use the **removeNode** command from the profiles' \bin directory.

   Note: The **removeNode** command only removes the node-specific configuration from the cell. It does not uninstall any applications that were installed previously as the result of executing an **addNode -includeapps** command. This is because the application or applications can subsequently deploy on other application servers in the cell.

   As a consequence, an **addNode -includeapps** command executed after a **removeNode** command was done does not move applications into the cell again because they already exist from the first **addNode** command. Thus, application servers added subsequently to the node do not contain any applications.

   To deal with this situation, add the node and use the Deployment Manager to manage the applications using the application administration panels or by adding new application servers to a cluster that hosts the application or applications.

3. On the **System administration** → **Nodes** panel, select all remaining nodes in the cell and click **Full Resynchronize** to make sure that all node's repositories are synchronized with the Deployment Manager.
4. Now, you can remove the hardware that previously hosted the wasna01 node from the network without impacting the WebSphere system. Remove the machine from the network.

4.3.2 Installing and configuring the new hardware or LPAR

We assume that the new hardware or new LPAR has the correct operating system level and any needed maintenance levels or fixpacks installed. (We refer to the new machine or LPAR as the *new system* for the remainder of this section.)

To see the latest list of IBM WebSphere Application Server Network Deployment V6 hardware and software requirements, visit:

http://www.ibm.com/software/webservers/appserv/was/requirements/

We reuse the same configuration as the replaced node (wasna01) had for our new system. Follow these steps to install and configure the new system:

1. Install the WebSphere Application Server Network Deployment software. You also need to re-install all refresh packs and fixes that were previously installed. WebSphere reinstallation can be done either manually or using a recent file system backup.

2. After successful installation of the core product files, you need to create a custom profile and federate the node into the cell.

   You can use either the Profile creation wizard or the **wasprofile** command to do so. Refer to Chapter 4, "Getting started with profiles" of *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451 for detailed information about profile management.

   Consider the following:

   – Use the original profile name and configuration settings from the removed node when creating the profile.

   – When using the Profile creation wizard you have the option to federate the node right away. However, you must be able to reach the Deployment Manager at this point, so the new system must already be in the network and the Deployment Manager must be up and running.

   – Alternatively you can federate the node after profile creation using the **addNode** *<dmgr_host>* *<dmgr_soap_port>* command. Section 5.5, "Working with nodes" of *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451 explains the **addNode** command in detail.

3. Federation of the node, using either one of the federation options, starts the Node Agent automatically so that you can use the Administrative Console for further configuration.

   Open the Administrative Console and verify that the Node Agent has indeed been started (**System administration** → **Node agents**). Alternatively you can use the **serverStatus nodeagent** command.

4. Recreate the cluster members wasmember01 and wasmember02 in wascluster01 and the cluster member wasmember05 in wascluster02. Cluster member creation is explained in great detail in Chapter 8, "Implementing the sample topology" of *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392.

   In the Administrative Console, go to **Servers** → **Clusters** and follow these steps:

   a. On the Server Cluster configuration panel, click **wascluster01** → **Cluster members** → **New**. Create two cluster members called wasmember01 and wasmember02 to the node wasna01.

b. Back on the Server Cluster configuration panel, click **wascluster02** → **Cluster members** → **New**. Add wasmember05 again to node wasna01.

c. Save your changes.

> **Note:** When a cluster member is created, WebSphere assigns new port numbers automatically for two WebContainer Inbound Chains called WCInboundDefault and WCInboundDefaultSecure. This setting is found under **Servers** → **Application servers** → ***App_Server_Name*** → **Web Container Settings** → **Web container transport chains**.
>
> If you did not use the default ports in your original configuration and would like to maintain the previously used port numbers for the newly created cluster members, be sure to go to this panel and make the appropriate changes (assuming that the original port numbers do not conflict with other ports on the new system). See Figure 4-5.

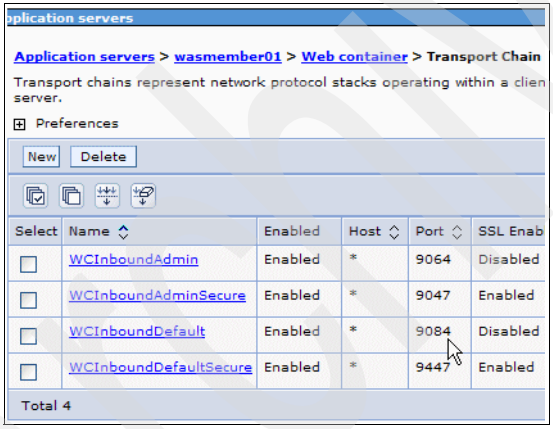


Figure 4-5 Transport Chain ports for cluster member wasmember01

5. Repeat any configuration changes that have been applied to the original cluster members, such as heap size and other individual settings. The new cluster members are created based on the configuration of the first cluster member and take their values from there.

6. Depending on your cell configuration, you might need to regenerate and propagate the Web server plug-in to pickup the new cluster members' configuration. This is needed if automatic generation and propagation of the plug-in is not enabled or if your Web servers are neither on a managed node nor an IBM HTTP Server v6 on an unmanaged node. See 15.3.2, "Web

server plug-in file (plugin-cfg.xml) management" on page 571 for more information about this topic.

To regenerate and propagate the Web server plug-in:

a. Select **Servers** → **Web servers**, select your Web server and select **Generate Plug-in**.

b. After regeneration, select the Web server and click **Propagate Plug-in** to ensure that the Web server has access to the most current copy of the plugin-cfg.xml file.

7. You are now ready to start the application servers on the new system. Go to **Servers** → **Application Servers**, select wasmember01**,** wasmember02, and wasmember05, and click **Start**.

8. Verify that you can indeed access the application on the added application servers by accessing the application directly through the WebContainer Inbound Chain of each cluster member.

4.4 Hardware upgrades

The different kinds of hardware upgrades have different impacts on your WebSphere environment. For example, if you are changing the disks of the machine and this requires reformatting the disks, then you need to follow the procedure described in 4.3, "Replacing hardware" on page 127.

For systems such as the IBM @server pSeries, iSeries, and zSeries, a disk capacity upgrade should not impact the running WebSphere processes because new disks can be added to the running system. Thus, they can be upgraded without stopping the node while the application is still available to users.

For other hardware upgrades, such as memory and processor upgrades, they are independent system processes and should not impact the configured WebSphere system, but might impact node availability itself for some platforms. A HA WebSphere system should always consist of multiple nodes so a failure or planned maintenance of one node does not impact the availability of the application. However, follow always the vendor's installation instructions to avoid unforeseen system problems.

Capacity on Demand (CoD) is the IBM solution designed to provide continuous availability for all major business applications, while providing dynamic system capacity growth and processor upgrade without causing hardware or software outages, activating and using the additional capacity are transparent to all

applications. To learn about IBM CoD, visit the *About Capacity on Demand* site available at:

http://www.ibm.com/servers/eserver/about/cod/about/types.html

For more information about CoD for the different IBM platforms, click the appropriate links on the *About Capacity on Demand* start page.

4.5 Installing WebSphere refresh packs

In this scenario we upgrade our nodes — one after the other — from IBM WebSphere Application Server Network Deployment V6.0.1 to V6.0.2 by installing the refresh pack using the Update Installer for WebSphere Software program. We refer to this program as Update Installer throughout this chapter.

The properties/version directory under the <*install_root*> contains data about WebSphere and its installed components, such as the build version and build date. You can use the **versionInfo** command to display the fix and version level of the various components (but do not use **versionInfo** *during* a product up/downgrade!). Search for *Product version information* in the WebSphere InfoCenter to learn more about this.

4.5.1 Downloading support packs

To download the latest WebSphere support packs for the Linux®, UNIX, Windows, and OS/400® platforms, visit the WebSphere Application Server support Web site at:

http://www.ibm.com/software/webservers/appserv/was/support/

The downloaded package normally contains both the Update Installer and the actual WebSphere maintenance package.

Each platform has its own prerequisites and requirements or known issues so make sure that you read the readme file for the appropriate platform before using the Update Installer. The readme also contains platform-specific installation instructions.

4.5.2 The Update Installer for WebSphere Software

The Update Installer updates the core product files of WebSphere Application Server and other related products, such as the IBM HTTP Server, the Web server plug-ins, the Application Client, and the WebSphere Edge Components.

The following files in the <*install_root*> directory might also be included in the update process:

- JAR files in the lib directory
- The SDK, Java technology edition, in the java directory
- Scripts in the bin directory
- Profile templates

Attention: There are some known problems and issues with the Update Installer. Refer to the WebSphere InfoCenter for the latest list and workarounds at the following Web address and search for *Update Installer*:

`http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp`

When installing a fix pack, refresh pack, or maintenance pack, make sure that you install the software as a user that has the needed authorizations to install the product updates, such as the root user on Linux and UNIX platforms.

Updating existing profiles

Maintenance packages do not update existing profiles. Thus, existing profiles might not have the latest features of the installed maintenance package.

Some maintenance packages provide *required service* for existing profiles in addition to service for the core product files. Each maintenance package that has profile maintenance provides a script that changes the profiles. The update installer prompts you to back up your configuration when installing a maintenance package that has required maintenance for profiles.

Some maintenance packages provide *optional service* for existing profiles. The readme file for the maintenance package describes whether the maintenance package contains optional service for existing profiles. If so, the readme file describes how to use the script provided with the maintenance package.

Cluster members

Apply the same maintenance packages to all WebSphere nodes in a cluster. When not all of the cluster members are at the same service level, an error can occur that causes memory-to-memory replication to not function properly. See Example 4-1 for the error message.

Example 4-1 Exception when cluster members are not all at the same service level

```
DRSCacheApp   E DRSW0008E:
   Exception is: com.ibm.disthub.impl.jms.JMSWrappedException:
   {-1361012295|unknown|java.io.OptionalDataException|}
```

4.5.3 WebSphere Application Server for distributed platforms

For the Linux, UNIX, and Windows platforms, see the *Applying service* article in the InfoCenter for additional platform-specific details on how to use the Update Installer:

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tins_ptfLevels.html

4.5.4 WebSphere Application Server for OS/400

WebSphere Application Server V6 for OS/400 software is upgraded using the Update Installer. However, it might be necessary to also install PTFs for Java, DB2, HTTP server, and so forth. Make sure that you read the iSeries specific Update Installer readme file as it lists required individual and Group PTFs.

The *Applying service* article describes how to use the Update Installer on an iSeries system and what user special authority is needed. You can find this article in the WebSphere Application Server for OS/400 V6 InfoCenter at:

http://publib.boulder.ibm.com/infocenter/wsdoc400/index.jsp?topic=/com.ibm.websphere.iseries.doc/info/ae/ae/os400_readme_updateinstaller.html

4.5.5 WebSphere Application Server for z/OS

The Update Installer is only used when upgrading WebSphere Application Server for z/Linux but not for WebSphere Application Server for z/OS®. WebSphere on Z/OS is installed via a SMP/E (System Modification Program Extended) process. SMP/E is an element of z/OS that is used to install most software products in z/OS and OS/390® systems and subsystems. It controls these changes at the element level by:

- Selecting the proper levels of elements to be installed from a large number of potential changes.
- Calling system utility programs to install the changes.
- Keeping records of the installed changes.

SMP/E is an integral part of the installation, service, and maintenance processes for z/OS and OS/390 software products and product packages. See the *Applying product maintenance* article in the WebSphere for z/OS InfoCenter for additional information. This article is available at:

http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp?topic=/com.ibm.websphere.zseries.doc/info/zseries/ae/tins_prodmaintenance.html

You can find the WebSphere Application Server for z/OS support Web site at:

http://www.ibm.com/software/webservers/appserv/zos_os390/support

4.5.6 Using the Update Installer

A few tips for using the Update Installer:

- It is important that you unzip or untar the update software into the <*install_root*> directory of the appropriate product. This unpacks the downloaded package automatically into the <*install_root*>/updateinstaller directory, which contains the Update Installer, and several subdirectories including the maintenance directory which contains the maintenance package itself. See Figure 4-6.

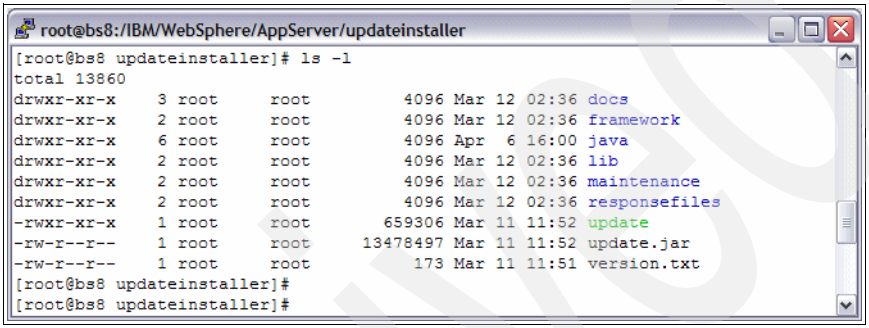


Figure 4-6 Subdirectories of the update installer directory

- It is very important that no WebSphere related processes are active on the system during the update or the installation fails. This includes not only the application server processes, Node Agent and Deployment Manager processes, but also the InstallShield and Profile creation wizard, the IBM Rational® Agent Controller process, or Application Client processes, and so on.

  If your update fails, you first need to uninstall the failed update package, then start the update again.

Example: installation on Linux or UNIX and Windows platforms

We assume that you have already downloaded the desired update package, such as WebSphere V6.0.1 or V6.0.2, from the WebSphere Application Server support Web site. We also assume that your environment matches the requirements for the update. For example, to install WebSphere V6.0.1.1, you must have 6.0.1 installed. You can install WebSphere V6.0.2 on top of version 6.0, 6.0.1, 6.0.1.1, or 6.0.1.2.

To install a WebSphere Application Server support pack:

1. Logon as `root` on the Linux or UNIX platform or as a member of the administrator group on Windows systems.

   For the Linux and UNIX platforms, verify that the umask setting is `022`. To verify the umask setting, issue the **`umask`** command. To set the umask setting to 022, issue the **`umask 022`** command.

2. Verify that there is enough disk space available. See the readme file for space requirements.

3. Backup and delete the existing <*install_root*>/updateinstaller directory before extracting the new .zip or .tar file. To use a newer version of the Update Installer, you must first remove the older version.

4. Extract the new package into the WebSphere <*install_root*> directory.

   Note: The unzip utility that is included with PKZip might not decompress the download image correctly. Use another utility (such as WinZip) to unzip the image.

5. Check the *Recommended updates* page (available from the WebSphere Application Server support Web site) to see if the Update Installer has been updated because the downloaded package file was created. If so, download and unpack the new Update Installer into the installation root directory to replace the Update Installer files that came with the package.

6. Run the **`backupConfig`** command to back up your configuration files. See the InfoCenter article *Backing up and restoring administrative configurations* or *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451 for details on this command.

7. Stop all WebSphere related Java processes.

   The update is very likely to fail if you install a refresh pack while a WebSphere related Java process is running. See Example 4-2 for the error.

Example 4-2 Error message when WebSphere Application Server processes are active

```
Failure: The update of the following product has failed:
IBM WebSphere Application Server Network Deployment - E:\IBM\WebSphere\ND

The following maintenance package could not be installed
PK00274 - Add jython.jar to ws_ant JVM classpath
For more information, refer to the following log file
E:\IBM\WebSphere\ND\logs\update\PK00274.install\updatelog.txt
Click Finish to exit the wizard
```

Tip: See the Technote *Stop all WebSphere Application Server-related Java processes before using the Update Installer for WebSphere software* for a list of processes that can cause the update to fail. The Technote is available at:

http://www.ibm.com/support/docview.wss?rs=180&uid=swg21199141

8. From the <*install_root*>/updateinstaller directory, invoke the **update** command to start the installation wizard.

Note: You can install updates also in silent mode. In this case, the Update Installer for WebSphere Software wizard reads the (optional) response options file to determine responses and does not display the graphical user interface. For example, run this command to use a response file called myresponsefile in the responsefiles directory:

./update -options “responsefiles/myresponsefile.txt” -silent

When you omit the response file, the default maintenance directory is used. For more information about the usage of a response file refer to the InfoCenter and search for *Install.txt*.

9. Refer to the updatelog.txt file under the <*install_root*>/logs/update/<*package_name*> directory for a complete listing of the components that have been successfully updated, as shown in Figure 4-7.

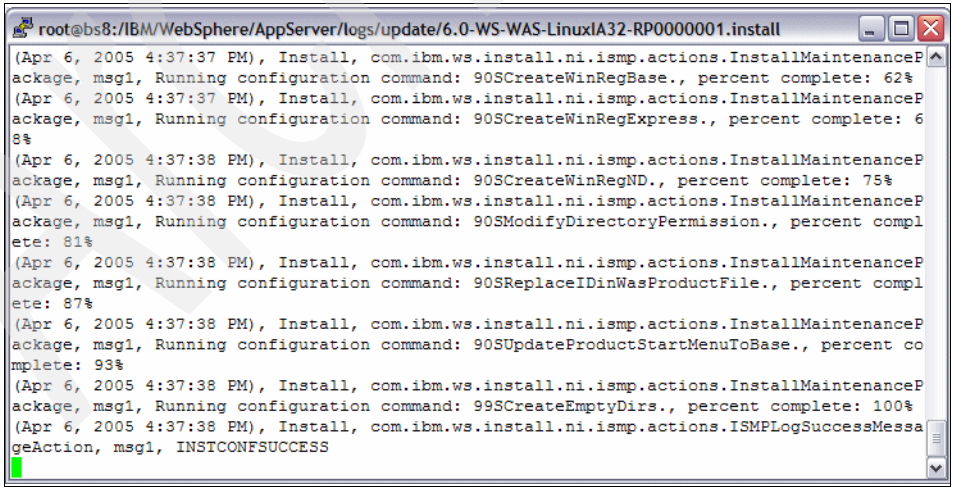


Figure 4-7 updatelog.txt

10. Restart the WebSphere processes on this node and update the next node to make sure all cluster members are at the same level.

4.6 Sample wsadmin scripts for administration tasks

The IBM developerWorks Web site provides many useful administration scripts (.jacl scripts) that you can use to configure and administer your WebSphere environment. You can run them “as is” or modify them for your own environment.

Note that some of the scripts are not yet available for WebSphere V6. Monitor the download site for future updates. You can download these scripts from:

http://www.ibm.com/developerworks/websphere/library/samples/SampleScripts.html

The available scripts include:

- Scripts for automated application deployment that perform functions such as:
  - Read a Distribution Directory dynamically to determine what EARs need to be installed or updated or reconfigured or uninstalled.
  - Use environment or stage specific property files to determine the target Nodes and Servers or Clusters and the application settings.
  - Calculate the unique set of affected Nodes and Servers (to avoid unnecessary interruptions to others).
  - Perform a phased update of affected Nodes and Servers to assist in maintaining high availability.
- Administration scripts that perform common administration functions, such as:
  - Create and modify a server, load an application onto the server, and start the server.
  - Stop a server on a given node, uninstall an application and remove the server from the configuration.
  - Invoke various application install commands.
  - Invoke commands that produce a short summary of configuration and runtime information about the WebSphere installation.
  - List all enterprise applications installed on a specific application server.

Note: At the time of writing this book, these administration scripts were available for WebSphere V5 but not V6.

- Scripts for WebSphere Application Server security configuration (V5, V5.1, and V6 versions available).
- Scripts for WebSphere Application Server configuration changes (V5.1.1 only).

5

High availability application administration

Having a highly available application server environment introduces certain issues when it comes to otherwise normal application administration tasks. For example, you must ensure that the application is available for your clients while performing tasks such as restarting the application or deploying a new application version. You cannot wait for the maintenance window, especially if your application or applications must be available 24 hours a day, 365 days a year, because it basically does not exist.

In this chapter, we explain the major issues regarding application restarting and updating. The underlying WebSphere topology is one of these issues. Another one is the type of application update to be performed. We distinguish between three different types of updates (major release, update, bugfix). This chapter also discusses application rollout update — a new function in IBM WebSphere Application Server Network Deployment V6.

5.1 Administering applications in an HA environment

Administering an application in a high availability (HA) environment has much more implications than in a standard environment. The goal when administering an HA application is to loose no requests while performing application maintenance tasks. Those maintenance tasks include:

- Restarting an application
- Deploying a new version of the application

Deploying an application for the first time is not covered here as it is out of the HA scope. For information about general application administration see Chapter 16 of the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

5.1.1 Availability while updating an application

It is not always well understood that updating an application can lead to situations where it is no longer available. Performing an update without paying special attention to things such as session replication, or even worse, restarting all application instances at the same time, might lead to downtime of the application or lost requests. The most obvious cause for downtime is a restart of an application or application server. Example 5-1 shows that the application was not available while restarting it.

Example 5-1 Restarting an application

```
[4/15/05 10:10:17:864 CDT] 0000012a ApplicationMg A   WSVR0217I: Stopping
application: BeenThere
[4/15/05 10:10:18:017 CDT] 0000012a EJBContainerI I   WSVR0041I: Stopping EJB
jar: BeenThere.jar
[4/15/05 10:10:18:028 CDT] 0000012a ApplicationMg A   WSVR0220I: Application
stopped: BeenThere
[4/15/05 10:10:24:646 CDT] 000000e4 ApplicationMg A   WSVR0200I: Starting
application: BeenThere
[4/15/05 10:10:24:702 CDT] 000000e4 EJBContainerI I   WSVR0207I: Preparing to
start EJB jar: BeenThere.jar
[4/15/05 10:10:24:717 CDT] 000000e4 EJBContainerI I   WSVR0037I: Starting EJB
jar: BeenThere.jar
[4/15/05 10:10:24:733 CDT] 000000e4 WebGroup      A   SRVE0169I: Loading Web
Module: BeenThere WAR.
[4/15/05 10:10:24:776 CDT] 000000e4 VirtualHost   I   SRVE0250I: Web Module
BeenThere WAR has been bound to
default_host[*:9080,*:80,*:9443,wasmember06.ibmredbook.com:9080,wasmember06.ibm
redbook.com:80,wasmember06.ibmredbook.com:9443,wasmember05.ibmredbook.com:9080,
wasmember05.ibmredbook.com:80,wasmember05.ibmredbook.com:9443].
[4/15/05 10:10:24:852 CDT] 000000e4 ApplicationMg A   WSVR0221I: Application
started: BeenThere
```

Another cause for unavailability are changes made to the application that interfere with HA services under the control of the HAManager. If a new application version has changed session objects signatures, you are exposed to class cast exceptions (ClassCastException).

5.1.2 System capacity

It is important that you know the workload during normal operation in your environment. Updating an application while keeping it available means that you need to run it on parts of your environment for a certain time.

For example, when it comes to an application rollout update, the application servers are stopped, updated, restarted on one node after the other. So for the duration of the update the remaining active servers must be able to handle the extra workload. This is a normal prerequisite for a highly available environment anyway but it might be more important during the application update because an entire node might not be available, while in a failure situation only one application server (one JVM) might fail.

The impact is most noticeable when using a two nodes topology because the computing resources might be cut in half during the update. So if each of your systems has equal processing powers and is using more than 50% of its capacity during normal operation, then the application update might lead to overloading the remaining system with side effects such as increased response times and maybe even request timeouts for the users.

5.2 Concepts

There are several concepts that you need to understand to ensure availability of an application during an update. Some of these concepts even have an implication on the application design. The two main concepts you need to understand are:

- Persistence layer
- Application update types

5.2.1 Persistence layer

The application depends on information storages managed by WebSphere Application Server. Modifying the way the application deals with them in a new application version might have consequences on availability.

The different types of information storage

As a service provider, WebSphere Application Server manages any kind of data that is outside of the user request scope, especially the following data types:

Session This is where the status of the user interaction with the application is kept. The session can contain any type of objects created by the application developers. Session data can be either replicated between the cluster members or written to a database.

Message Application processes can use messages to communicate with each other. The message queue is independent of the application itself.

Transaction Ensures that an interaction initiated by a user has terminated in a predictable manner. Failover of a transaction will in most cases be handled by another application server JVM which possibly runs a different version of the application.

Application and persistent information interaction

WebSphere Application Server ensures that data is not lost after a failure of any of its components. For example, it replicates the session object data between all (or a defined number of) cluster members or stores the session data in an external session database when distributed session management is enabled so another application server can take over the request.

Attention: If your application must be available at all times, even during an application update that includes changes to persistent data, then either the developer must take special precautions in the new application version or you need a specific architecture to ensure availability during the update. Architectures are described in 5.3, “Topologies” on page 146.

The following is a brief description of how an application should handle persistent data to avoid inconsistencies:

- When session data changes between application versions, then the new application version should still be able to read and process sessions from the old application version or you must make sure that all sessions from the old version expire before updating the application (see “Tracking live sessions” on page 157).

Let us look at an example: You are adding a *Preferred language* indicator to the session data. A session was created in the old application version without the Preferred language indicator. The application should not end with an error because of the missing information but rather the new application version should be designed to handle this situation. For example the existing session data could be copied into a new session object that includes a default value for the preferred language.

- If an old application version process sends a message that will be consumed by a newer application version process, then the new version should either expect the same message than the previous version used, or the new version must be designed to handle the different message format.
- If an unfinished transaction fails over to a newer version process, the application should handle the transaction exactly like the previous version or again be designed to handle the situation with the different transaction data.

5.2.2 Application update types

We distinguish three different types of application updates - depending on the impact they have on WebSphere managed information as described in “Persistence layer” on page 144. These update types are: major release, update, and bugfix.

Major release

Major release updates involve back-end application modifications, for example, the application database schema needs changes, a Web Service from a third-party provider changed or a back-end application is modified which has then an impact on the WebSphere application. So a major release update involves changes outside of the WebSphere environment.

Upgrade

Changes to an application that have an impact on the persistence layer of WebSphere Application Server are considered an application upgrade. There are two possibilities:

1. Your application handles these changes gracefully. In this case, you can update the application regardless of your WebSphere topology.
2. Your application needs to have only new version objects in the persistence layer. You then need a specific environment for the update. See 5.3, “Topologies” on page 146 for information about the various possible environments.

Important: WebSphere Application Server cannot validate changes in the application. Thus, it is *your responsibility* to determine whether your application deals gracefully with the persistence layer before performing the application update!

Bugfix

This kind of update does not modify any critical resources, such as session objects or database schemas. This is the update type with the lowest impact on runtime resources. Many bugfix updates can even be made with a running application server (hot deployment). For more information about replacing single files in an application or module, see Chapter 16 of *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

5.3 Topologies

How you manage your applications also depends on your WebSphere topology. Refer to 1.2.6, “Levels of WebSphere system availability” on page 18 for additional information about WebSphere HA levels. For an application update, we distinguish between the following possible topologies:

- Multiple cells environment
- Single cell, multiple clusters
- Single cell, single cluster

Note: We assume a minimum of two HTTP servers and a network sprayer (we use the WebSphere Edge Components’ Load Balancer throughout this chapter) for all our scenarios. Having only one HTTP server would introduce a single point of failure. In addition, updating static content on the HTTP server might become an issue if there is only one HTTP server.

5.3.1 Multiple cells environment

This is the top level HA topology, which addresses even a site disaster recovery. It is the most complex environment to manage, with repetitive administration tasks for each one of the cells. On the other hand, it is also the most flexible environment and allows you to acquiesce a full cell at a time without having any availability issues.

Figure 5-1 on page 147 shows this architecture.

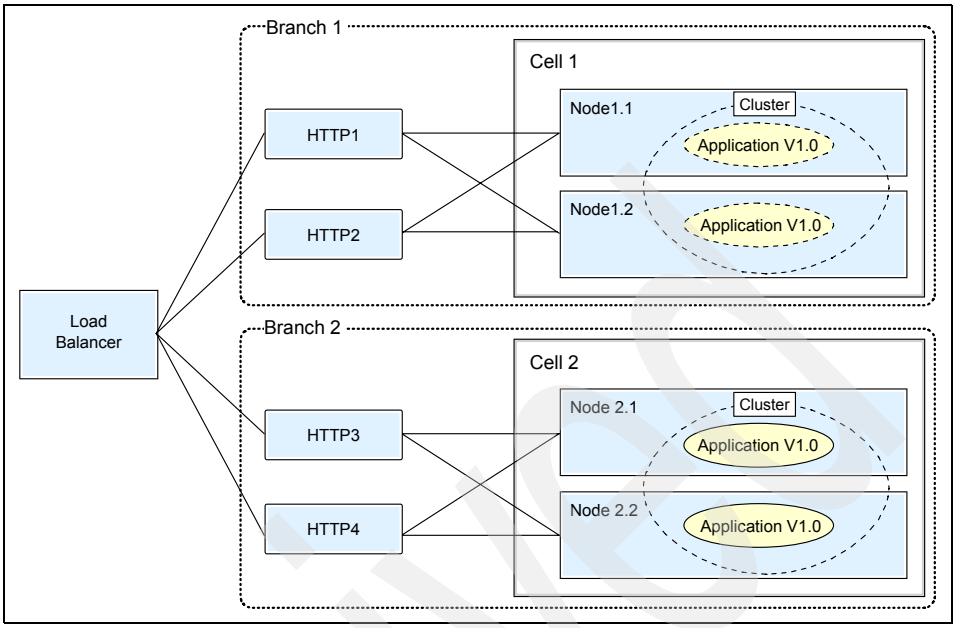


Figure 5-1 WebSphere topology with multiple cells

Note: For the scenario depicted in Figure 5-1, server affinity must be enabled at the Load Balancer to make sure that subsequent requests are routed to the same cell. Another option is to use two different URLs to access the different cells.

Any variation of this topology can be used, as long as the following steps can be performed:

1. Isolate a branch (cell).
2. Wait for all sessions on this branch to expire.

Important: You have to wait for the sessions to end, because there is no replication between the cells and thus no failover.

3. Update the offline branch.
4. Make the updated branch available again. This branch now serves the new application version while the other branch keeps on serving the old application version.
5. Repeat steps 1 to 4 for the second branch.

5.3.2 Single cell, multiple clusters

This topology is very similar to using multiple cells, so the considerations stated in 5.3.1, “Multiple cells environment” on page 146 also apply for this scenario. For a single cell, multiple clusters topology the isolation of a branch means the isolation of a cluster.

Important: This topology has a major drawback and, thus, is *not* a recommended approach. In order to have the same application running on two clusters in one cell, you actually need two differently deployed instances of the same application. It is very important that your application design or development allows for this. For example, there must be no hardcoded information, such as JNDI references, in the application.

Deployment is more difficult for the second instance of the application, several modifications must be made. For example, for the second installation, you need a different context roo and different resource mappings (EJBs, datasources, and so forth). You might also need to use different virtual hosts.

Because it is possible to configure multiple cells on only two physical machines, we strongly recommend that you use the multiple cells approach rather than the multiple clusters approach.

Depending on your topology, you might be able to achieve the isolation of one cluster at the Load Balancer level or at the Web server plug-in level. See “Updating a WebSphere application and static content on the HTTP server” on page 168 for more information about how to edit the plugin-cfg.xml file to isolate application servers.

Figure 5-2 shows such a topology.

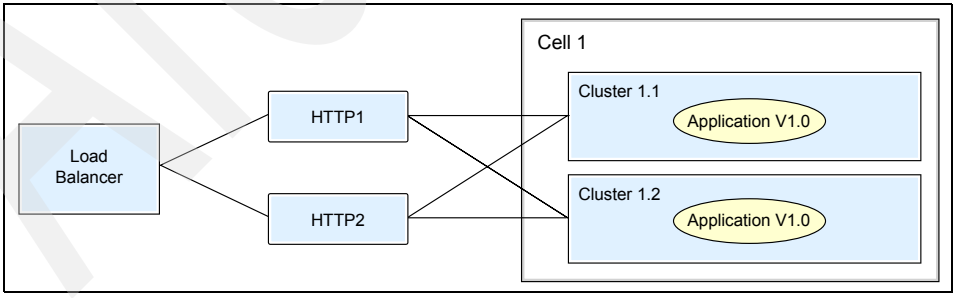


Figure 5-2 WebSphere topology with a single cell and multiple clusters

The high-level approach for an application update is as follows:

1. Make sure no new requests are sent to the first cluster (for example by setting the application server's weights to 0). Wait until all active sessions have expired. This is needed because there is no failover from one cluster to the other.
2. Stop the cluster when all requests are finished.
3. If you need to change static content that is located on the HTTP server: quiesce, then stop one HTTP server at the Load Balancer level.
4. Update the application on the stopped cluster and update static content on the stopped HTTP server.
5. Make sure the updated HTTP server only sends requests to the updated cluster (for example by mapping only this HTTP server to the application using the Administrative Console or by changing the plugin-cfg.xml file).
6. Restart the HTTP server and application server cluster.
7. Repeat steps 1 to 6 for the second cluster and HTTP server.
8. Change the application modules mapping/plug-in file back to its original state so both HTTP servers can send to both clusters again.

5.3.3 Single cell, single cluster

This is the most simple topology to administer and ideal for an environment that accepts downtime while updating an application. The downtime is related to the fact that there is no option to isolate a branch for the update.

If the cluster members are on different nodes (horizontal scaling), then this topology also allows for an application rollout update which means that the different nodes are updated one after the other while having the application up and running on the other node or nodes. However, the rollout update is not always the right solution, for example, when it comes to a major release update with changes to the back-end data or if your new application version cannot handle changes to the persistence layer gracefully. See 5.4.2, "Rollout update (new feature of WebSphere V6)" on page 153 for more information about this function.

Changes to the cluster configuration or application must always be handled with care, but this is especially true when having only one cluster available because there is no other cluster and thus application version available in case something goes wrong.

Figure 5-3 on page 150 shows the single cell, single cluster topology.

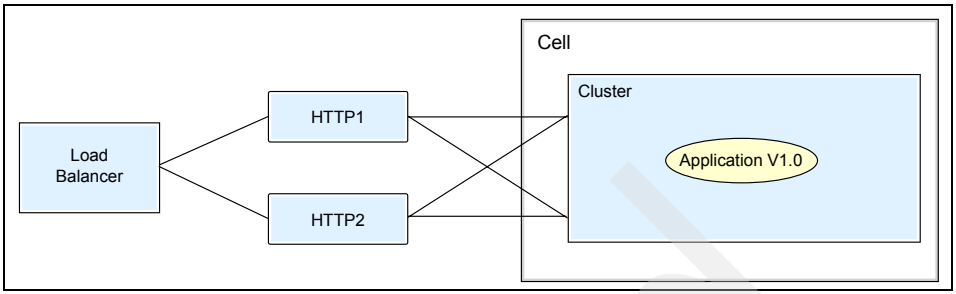


Figure 5-3 WebSphere topology with a single cell and single cluster

5.3.4 Topologies and update types

The ability to keep an application available during an update is influenced by the factors described earlier in this section, such as the topology and the application update type. But the contributing factor is that your topology is compatible with the update type of the application. For example, when your update is a major upgrade, then you must have either a multiple cell or at least a multiple clusters topology.

Table 5-1 details which application update types can be performed on each topology while keeping the application available.

Table 5-1 Application availability depending on topologies and application update types

	Multiple cells / multiple clusters	**Single cell, single cluster**
Major release	OK	Not possible[a]
Upgrade	OK	Possible (under certain circumstances and using special precautions)[b]
Bugfix	OK	OK

a. The major release update involves changes in the EIS tier. You cannot have different WebSphere application versions in one cluster accessing different data or back-end applications.
b. If your new application version can handle changes to the persistence layer gracefully.

For more information, see 5.4.3, “Update types: major release or upgrade” on page 156 and 5.4.4, “Update type: bugfix release” on page 164.

5.4 Application administration

In this section, we explain how to restart and update applications in an HA environment, including the new application rollout update function. We explain step-by-step how to update an application in a multiple cell and in a single cell, single cluster topology respectively. We cover the following scenarios:

Prerequisites

We assume the following underlying WebSphere configuration:

- At least two cluster members are running on two different nodes
- At least two HTTP servers and a Load Balancer are available
- A HA solution for the Transaction Manager is set up (HAManager)
- A HA solution for messaging is set up (HAManager)
- Distributed session management is configured and active (memory-to-memory or database persistence)

5.4.1 Restarting an application

If for any reason you need to restart an application, the best way to keep the application available during the restart is to stop and start, one after the other, all application servers the application is deployed to.

This can either be done manually or by using the Ripplestart option on the cluster management panel (**Servers** → **Clusters**) in the Administrative Console. The Ripplestart automatically stops and restarts each application server in the cluster, one after the other.

Important: We do not recommend to use the Ripplestart option in an environment with only a few cluster members (especially with only 2 members) and when 100% application availability is important. Using the Ripplestart in an environment with only a few cluster members can lead to Internal Server Errors and failed requests. The reason for this is the way in which the Ripplestart works:

1. The first cluster member is stopped. It is marked down by the Web server plug-in. All requests go to the second cluster member.
2. The first cluster member is restarted and back up.
3. The second cluster member is stopped and marked down by the plug-in. All requests flow to the first cluster member.
4. The second cluster member is restarted and the workload is distributed between the cluster members again.

The problem is found in step 3. When a cluster member is marked down, the plug-in will not try it again before the retry interval has passed (see “Retry interval” on page 57). If now the second cluster member is stopped before the retry interval of the first cluster member has passed, the plug-in will not try the first member, even though it is up again, and thus cannot find an active member to forward requests to.

This is most probably not an issue when you have many cluster members and your retry interval setting is not particular high. See 4.2, “Starting or stopping application servers and the Web server plug-in retry interval” on page 125 for details on the correlation between the retry interval and starting/stopping application servers.

Thus, whether you are able to use the Ripplestart option depends on the number of application servers (and their capacity) and the retry interval setting in your environment.

As mentioned, the stopping and starting can be done manually. However, a better solution is to write a script that performs the stops and starts using **`wsadmin`** commands. Using such a script you can add a delay between the start of one cluster member and the stop of the next one that takes your settings for the retry interval (the default is 60 seconds) into account.

Note: During our tests, we noticed that we lost a few in-flight HTTP requests when stopping a cluster member. This problem has been fixed and requires WebSphere V6.0.2.1.

5.4.2 Rollout update (new feature of WebSphere V6)

A new feature of IBM WebSphere Application Server Network Deployment V6 is the application rollout update which allows to sequentially update an application on different nodes. So, you can only use this function if your cluster members are configured on at least two nodes (horizontal scaling).

Note: This function requires at least WebSphereV6.0.1.2 or V6.0.2.

The rollout update is a good solution for bugfix type application updates. Upgrade type updates can also use the rollout update function if the application is able to handle changes to the persistence layer gracefully, as explained in 5.2.1, "Persistence layer" on page 144. If this is not the case, then you need to use the application update approach documented in 5.4.3, "Update types: major release or upgrade" on page 156.

Attention: Also for the rollout update, you need to be aware of the correlation between the Web server plug-in retry interval and the time it takes to stop and restart the application servers for the application update. If your retry interval is too high, then the rollout update might lead to failed requests, especially in an environment with only two nodes. Before using this function, you should, therefore, verify your configuration settings and test the rollout update in a meaningful non-production environment.

See 4.2, "Starting or stopping application servers and the Web server plug-in retry interval" on page 125 for details.

How to perform a rollout update

To perform a rollout update, you start with a classical application update, but then you perform an additional step rather than saving the update to the master repository. Follow these steps:

1. In the Administrative Console, go to the application administration panel. Click **Applications** → **Enterprise Applications**.
2. Select your application and click **Update**. Do not use Rollout Update at this point. It is not useful until you have an application update pending, which is initiated by clicking **Update**.
3. On the Preparing for the application installation panel, select either **Local file system** or **Remote file system** radio, depending on where your new application version is stored. Then click **Browse...** to select your .EAR file. See Figure 5-4 on page 154.

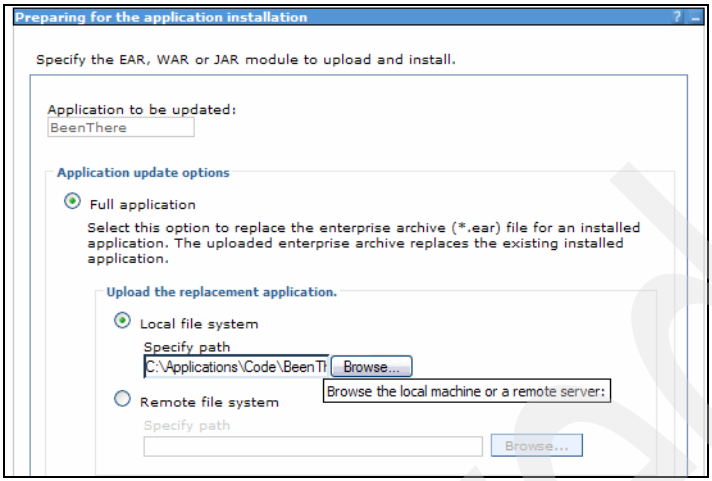


Figure 5-4 Application update - select the .EAR file

4. Click **Next** two times. In case the Application Security Warnings screen appears, click **Continue**. You should now be on the Install New Application panel. Click the **Step 2** (Map modules to servers) link.
5. Verify that your application is mapped correctly to the appropriate target cluster and that the Web modules are also mapped to at least one HTTP server. This is shown in Figure 5-5 on page 155.

Restriction: The rollout update only works for applications that are deployed to a cluster. Obviously, a rolling update cannot be performed on a single server.

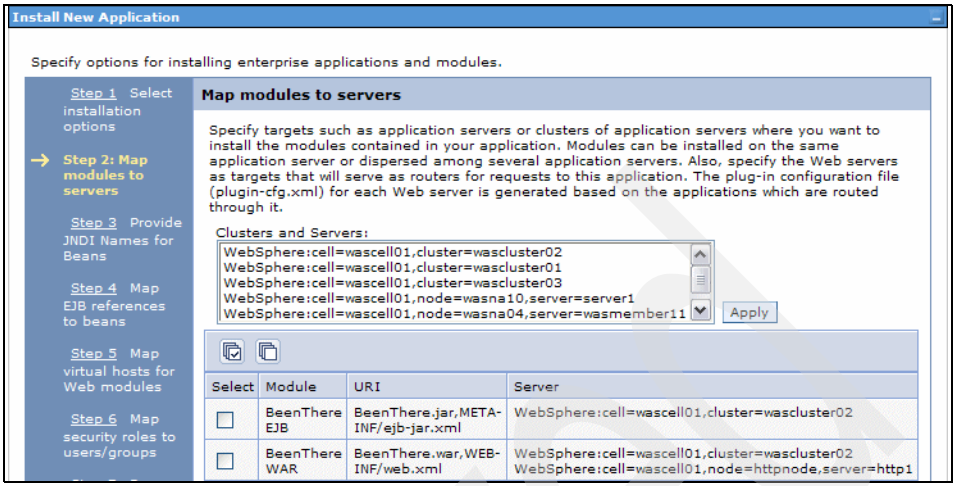


Figure 5-5 Application to cluster mappings

6. Make all other necessary deployment changes, such as mappings for EJB references or the virtual host. These changes depend on your application. You can either step through the remaining configuration steps using Next or by directly clicking the appropriate Step links. Finally, you should arrive at Step 8, Summary.
7. Click **Finish**. The panel shown in Figure 5-6 is displayed.

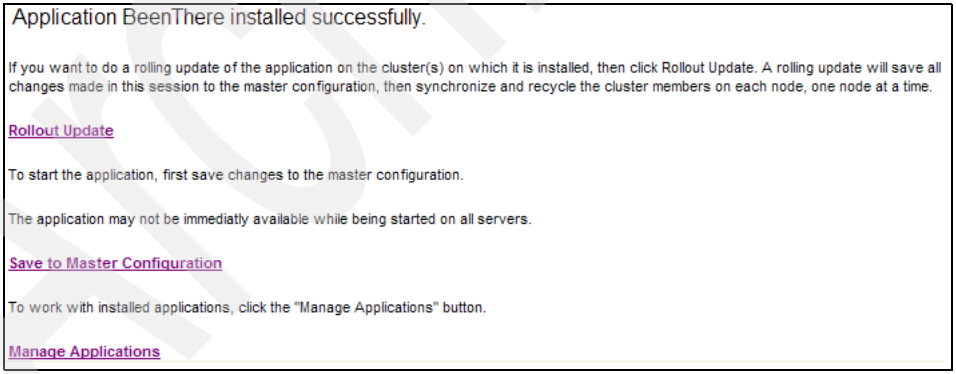


Figure 5-6 Application installed successfully

8. You now have two choices, both lead to the same result:
   a. Click the **Rollout Update** link.
   b. Click the **Manage Applications** link, then select your application from the Enterprise Applications panel and click **Rollout Update**.

> **Important:** Do *not* select the **Save to Master Configuration** link!

9. If the Confirm Application Rollout Confirm panel appears, select **Yes**.
10. You then see a scrolling log of the rollout update as shown in Figure 5-7.

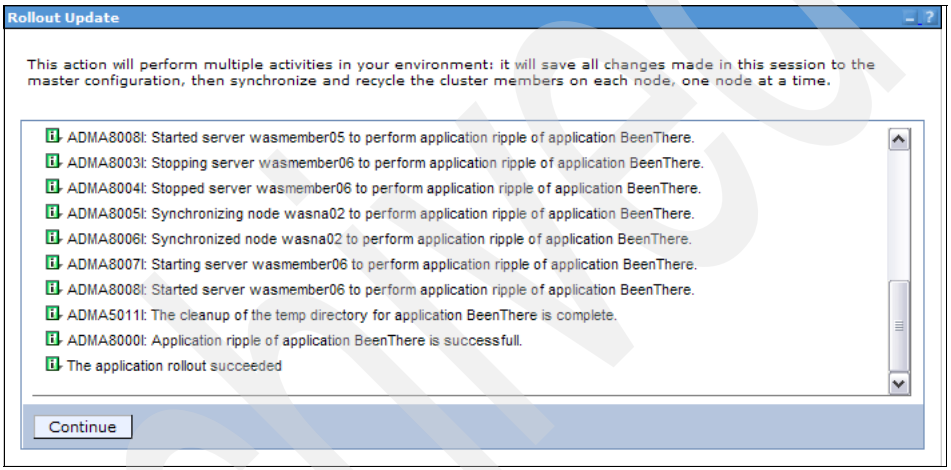


Figure 5-7 Application rollout - success message

> **Note:** Do not refresh, reload, or press F5 on the Administrative Console during the application rollout update, this might cause inconsistencies between the different nodes!

5.4.3 Update types: major release or upgrade

As seen in Table 5-1 on page 150, you need at least a multiple cluster topology to achieve a major release update while keeping the application available during the update. This is because a major release update involves changes in the EIS tier, either the database or in a back-end application.

An upgrade type application update can be done in a single cell environment if the application is able to handle changes to the persistence layer. If this is not the

case, then you need at least a multiple cluster topology to stay online with your application while updating it.

Another issue for these two update types might be that you have to wait for active sessions to expire. This is true for the multiple cells and multiple clusters topologies because there is no failover between cells or clusters. Therefore the first step during the update is to make sure no new requests arrive at the cell or cluster.

Tracking live sessions

Important: As mentioned earlier, there is no replication or failover between different cells. Therefore, before stopping a cell, you must be sure that there are no more live sessions in the cell. In other words, you must ensure that there are no users actively using the application in the cell or their requests will be lost.

One way to check that there are no more active sessions in the system is to use the Tivoli Performance Viewer (TPV). In the Administrative Console, navigate to **Monitoring and Tuning** → **Performance Viewer** → **Current Activity**.

Click the link for your first cluster member from the list of application servers. Expand **Performance Modules**, and then check **Servlet Session Manager**.

Figure 5-8 on page 158 shows the needed Tivoli Performance Viewer settings in order to display the live sessions count.

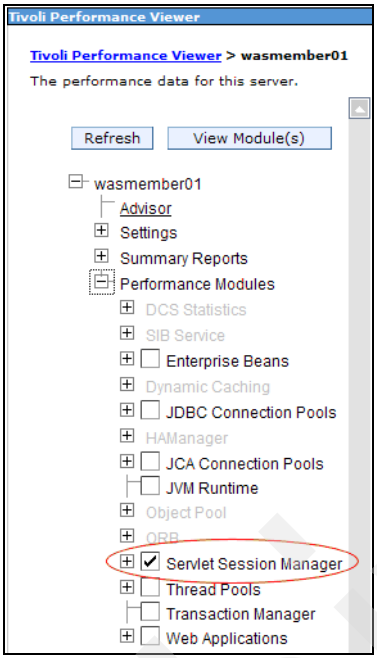


Figure 5-8 Module selection to show sessions count in Tivoli Performance Viewer

When this is enabled, the pane on the right hand side, shown in Figure 5-9 on page 159, shows the current live sessions in that application server. Typically, you see this graph after a certain amount of time.

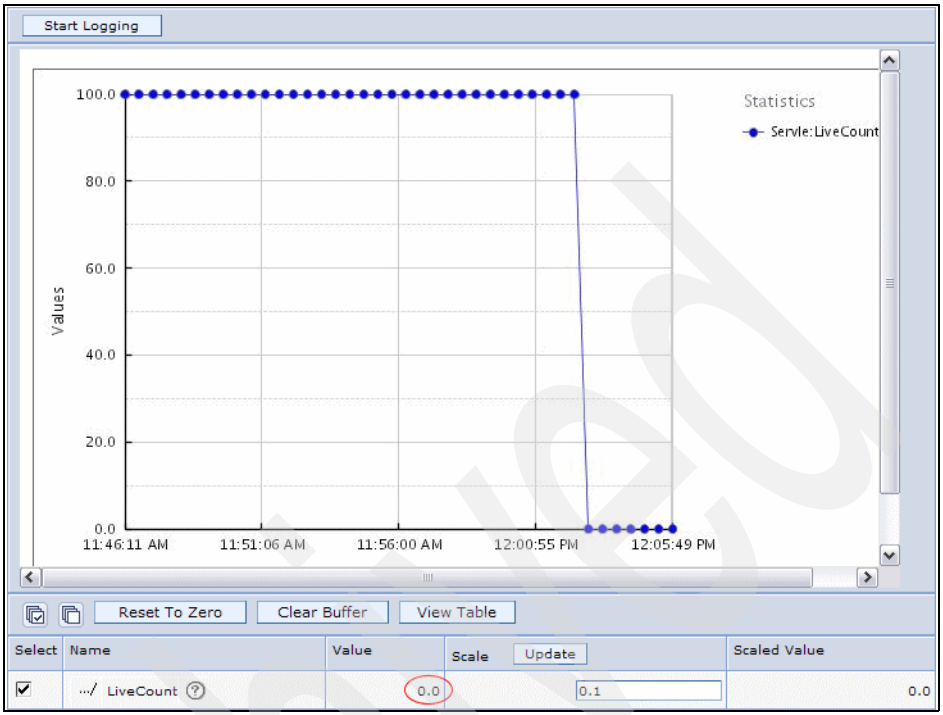


Figure 5-9 The number of live sessions is now 0

As you can see in the bottom line, the LiveCount metric is now 0, meaning that you do not have any more live sessions on that specific cluster member.

Important: You need to repeat this action for each cluster member that is to be stopped and updated.

The update scenario

As mentioned in 5.3, “Topologies” on page 146, you should isolate a branch of your environment and then update this branch. A branch can be an entire cell (in a multiple cells environment) or a cluster (in a multiple cluster topology).

Note: Even some individual cluster members (in a single cell, single cluster topology) might be possible to isolate. See 5.4.4, “Update type: bugfix release” on page 164 for information about how to do this. However, this does not make sense for major release updates and for upgrade type application updates it can only be used when the application is able to handle changes to the persistence layer as explained in 5.2.1, “Persistence layer” on page 144.

Considerations for updating a multiple cells topology

When using the same URL to access the application in both cells and the Load Balancer distributes the workload between the two cells (this is the topology shown in Figure 5-11 on page 163), then you need to enable session affinity in the Load Balancer to make sure subsequent requests are routed to the same cell. You must then quiesce and stop the HTTP servers that serve the requests to the first cell as explained in step 3 on page 161 below.

Considerations for updating a multiple clusters topology

We assume that all HTTP servers in the environment can send requests to all application servers in all clusters. See Figure 5-10 on page 161.

If static content on the HTTP servers does not change, then you can simply take the cluster to be updated offline (as explained in steps 1 on page 161 and 2 on page 161 below) and all HTTP servers can continue sending requests to the other cluster or clusters.

If static content on the HTTP servers needs to be changed and thus an HTTP server needs to be taken offline during the application update, then you first need to *untangle* the HTTP server to application mapping to make sure you have a fixed assignment between one or more HTTP servers and the cluster (meaning: application) you want to update. This can either be done by changing the application module mapping using the Administrative Console or by editing the plugin-cfg.xml file. The next step is then to quiesce and stop the HTTP servers that serve the requests to the first cluster as explained in step 3 on page 161 below.

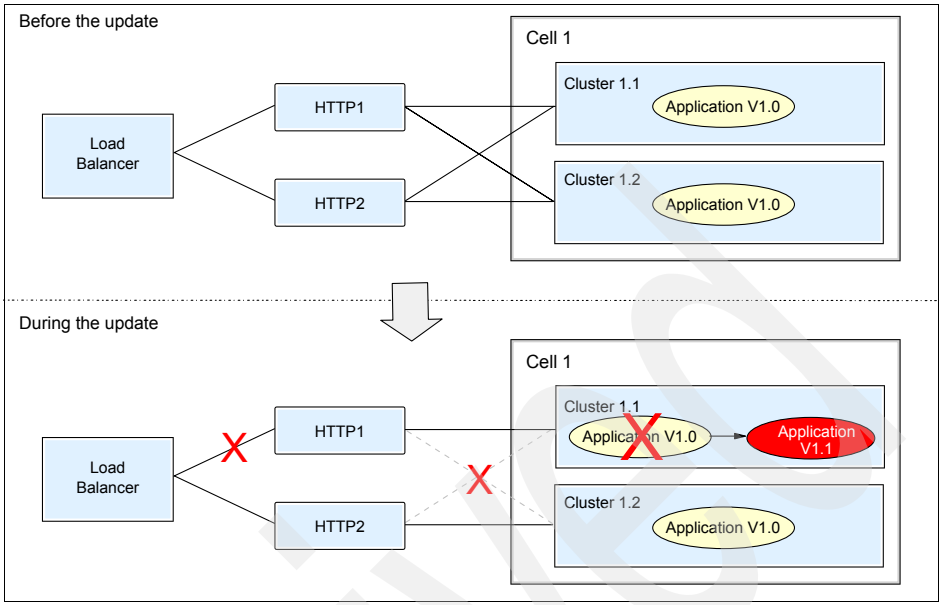


Figure 5-10 Multiple clusters - assign one HTTP server to one cluster during update

The update steps

The main steps for updating a branch are:

1. Set the application server weights to 0 to make sure no new requests arrive at the cell or cluster. Affinity requests are still routed to the servers in this branch. This can be done on the Runtime tab of the cluster member configuration panel. Go to **Servers** → **Clusters** → ***Cluster_name*** → **Cluster members** → ***Member_name***. Select the Runtime tab and enter 0 into the Weight field. This change becomes active as soon as you click **Apply** or **OK**. The original weights are used again when the cluster is restarted after the application update.
2. Monitor the live sessions of all application servers on this branch. See “Tracking live sessions” on page 157 for additional information. Wait for all sessions to be either closed or timed out. This ensures that there are no more active transactions, as nobody is connected to the servers any more.
3. In case you need to update static content on the HTTP server or servers, quiesce the HTTP servers at the Load Balancer and wait for a reasonable amount of time so all affinity requests to the HTTP servers have finished or expired.

   For the Load Balancer, you can use the **`dscontrol manager quiesce`** command. The **`quiesce`** subcommand allows existing connections to complete (without being severed) and forwards only subsequent new

connections from the client to the quiesced server if the connection is designated as sticky and stickytime has not expired. The quiesce subcommand disallows any other new connections to the server. You can check the number of connections to that server with the **`dscontrol server rep ::`** command. When there are no more active connections, you can safely shut the server down. See the Load Balancer Administration Guide Version 6.0 for more information about these commands.

4. Stop the application servers in the cell or cluster. Stop the HTTP servers if static content is to be updated.
5. Update the application. If you need information about how to update an application see *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

   Depending on the update type, do one of the following:

   – For a major release update, first update the back-end system (database, messaging, middleware, and so forth), and then update your application.

   > **Note:** We do not explain any details on what might be necessary to update anything that is outside of the WebSphere cell, such as a database schema, and so forth. Make sure that you know exactly what you are doing and what the impact on the overall system might be.

   – If only the application needs to be updated, update your application.
6. Start the cell or cluster.

   Verify that your new application version works properly before going back online with this branch. Figure 5-11 on page 163 shows the actual state just before switching the updated branch back online in a multiple cell topology. The Load Balancer is not routing any requests to the first branch but the application is already updated to the new version and can be tested on the intranet.

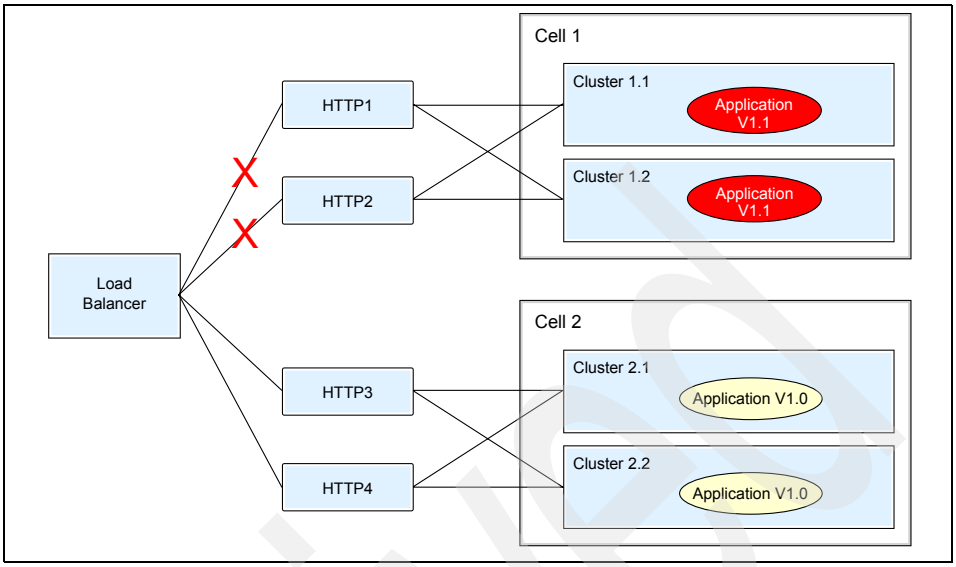


Figure 5-11 Multiple cells: during the update, phase 1

7. After updating and testing the first branch you can then reactivate it at the Load Balancer level.

   Figure 5-12 on page 164 shows the state between reactivation of branch 1 and starting to quiesce branch 2 for a multiple cells environment. Both branches are active, however, with different application versions.

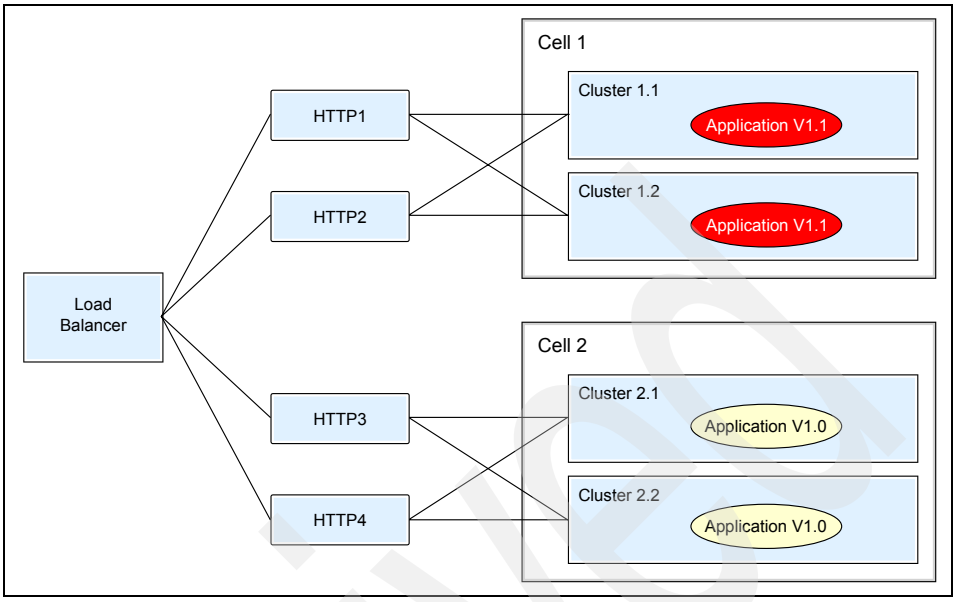


Figure 5-12 After branch 1 has been updated - both application versions are online

8. Repeat steps 1 on page 149 to 7 on page 163 to update the next branch.

5.4.4 Update type: bugfix release

This is probably the most common and thus most interesting update type. It can be performed on a running system for all topologies. Using the rollout update function, as described in 5.4.2, “Rollout update (new feature of WebSphere V6)” on page 153 is also an option for this type of application update if your environment allows for it.

The minimum configuration to achieve this without application downtime is a cluster on two different nodes (horizontal scaling). Each node hosts at least one cluster member. If you need to update static content on the HTTP servers also, then you need a minimum of two HTTP servers.

In this scenario, you might not need to wait for active sessions to expire or end, as they can failover to the other, still active, cluster members. So, a bugfix update is a much faster overall process.

The topology that is described in 5.3.3, “Single cell, single cluster” on page 149 is shown in more detail in Figure 5-13 on page 165. We now need to take the cluster members and the Web server plug-in configuration file into account.

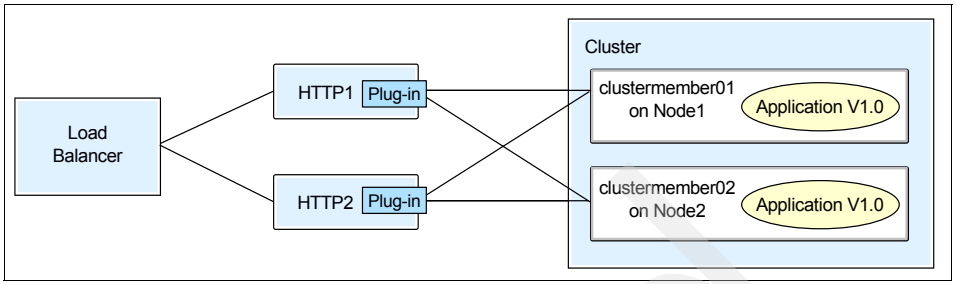


Figure 5-13 Single cell, single cluster topology - the details

The update scenario

The update scenario depends on whether you also need to update static content on the HTTP servers or only the WebSphere application. In both cases, however, you first need to deactivate the automatic file synchronization service of the Node Agents. Next, you update the WebSphere application and also any static content on the HTTP servers that needs to be updated.

After deactivating the file synchronization service, follow “Updating a WebSphere application only” on page 166 if no static content needs to be updated. Alternatively, follow “Updating a WebSphere application and static content on the HTTP server” on page 168 if you need to update static content on the HTTP servers.

Deactivating the automatic file synchronization service

The following is a step-by-step description of this process:

1. Using the Administrative Console, navigate to **System administration** → **Node agents**.
2. Click **nodeagent** to open the configuration page, and then select **File synchronization service**.
3. Deselect (the fields **Enable service at server startup** and) **Automatic synchronization** as shown in Figure 5-14 on page 166.

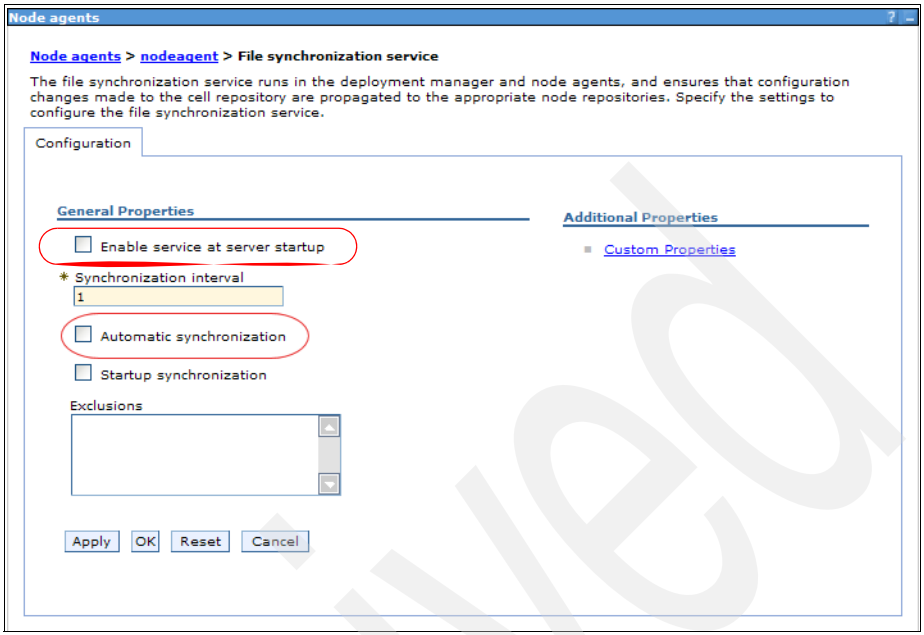


Figure 5-14 Disable Automatic synchronization

4. Repeat these steps for each Node Agent in the environment.
5. Save your configuration changes, and propagate them to the nodes using **Synchronize changes with Nodes** on the save page.
6. Restart all Node Agents. You can use the Administrative Console Restart.

Updating a WebSphere application only

If you need information about how to update a WebSphere application, see *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

Attention: Follow these steps only if *no static content* needs to be updated on the HTTP servers! Follow the instructions in “Updating a WebSphere application and static content on the HTTP server” on page 168 if you need to update the WebSphere application as well as static content.

To update a WebSphere application:

1. Update the application and save it to the master configuration. Be careful *not* to synchronize with the nodes during this task.

   At this point the updated application resides in the master repository on the Deployment Manager system. The nodes and thus cluster members are not aware of the new code version and continue to serve the old application version that is stored in their local configuration repository subset.

2. Next you synchronize one node after the other using the Administrative Console. Go to **System administration** → **Nodes**. Select the first node and click **Full Resynchronize**. See Figure 5-15.

   The synchronized node now serves the new application version while the other nodes continue to serve the old application version. As a precaution you can test your application on the updated node before synchronizing the other nodes.

3. Synchronize your other nodes accordingly. When all nodes are synchronized, all of them serve the new application version.

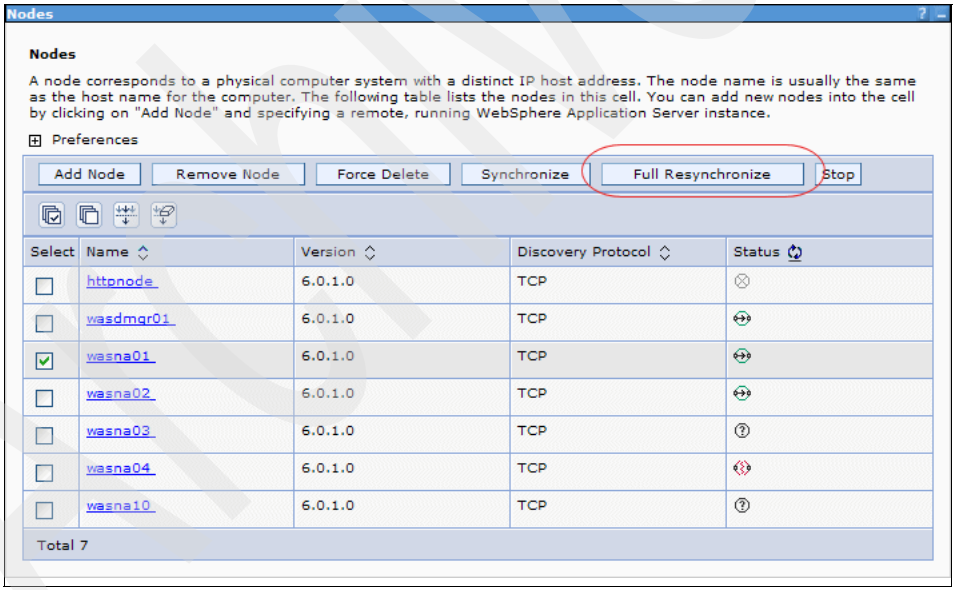


Figure 5-15 Resynchronize the node

4. After updating the application on all nodes you can re-enable the file synchronization service for the Node Agents. Follow the steps described in “Deactivating the automatic file synchronization service” on page 165 but this time check the two fields. Do not forget to restart the Node Agents.

Updating a WebSphere application and static content on the HTTP server

In order to update static content on the HTTP servers, you must be able to remove the HTTP server from the environment while updating it. This means that you need a minimum of two HTTP servers for this to work seamlessly. In most cases, all HTTP servers are associated to all cluster members and can thus forward requests to the entire cluster - this is shown in Figure 5-13 on page 165. Thus, for this update scenario, you must first untangle the HTTP server to application mapping to make sure that each HTTP server is only associated with certain cluster members (we call this a branch again throughout this section). This situation is depicted in Figure 5-16.

Note: All cluster members on a node must belong to the same branch but you could have more than one node in a branch.

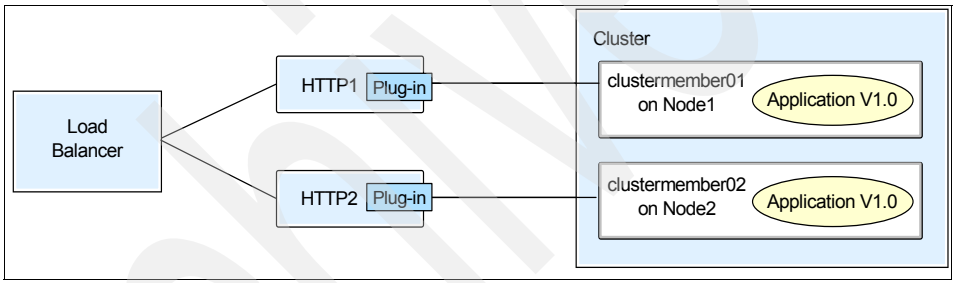


Figure 5-16 HTTP server and nodes: one-to-one assignment

Important: There is no failover of active requests available any more when the branches are created because the HTTP server does not know about the other application servers. Therefore you must make sure that there are no active sessions in the branch you wish to update.

Follow these steps to create and update the branches:

1. For the application servers of the first branch: Stop the application server so that active requests failover to the application servers of the other branch. Alternatively you can set the application server weights to 0 to make sure no new requests arrive and monitor for active sessions to end as described in “The update steps” on page 161 before continuing with creating the branches.
2. Using the Administrative Console, deactivate automatic propagation of the plug-in file to the HTTP server. Click **Servers** → **Web servers** → ***Web_Server_Name*** → **Plug-in properties**. Deselect **Automatically**

propagate plugin configuration file on the configuration panel shown in Figure 5-17. Click **OK** and save your changes.

Do this for all HTTP servers in your environment.

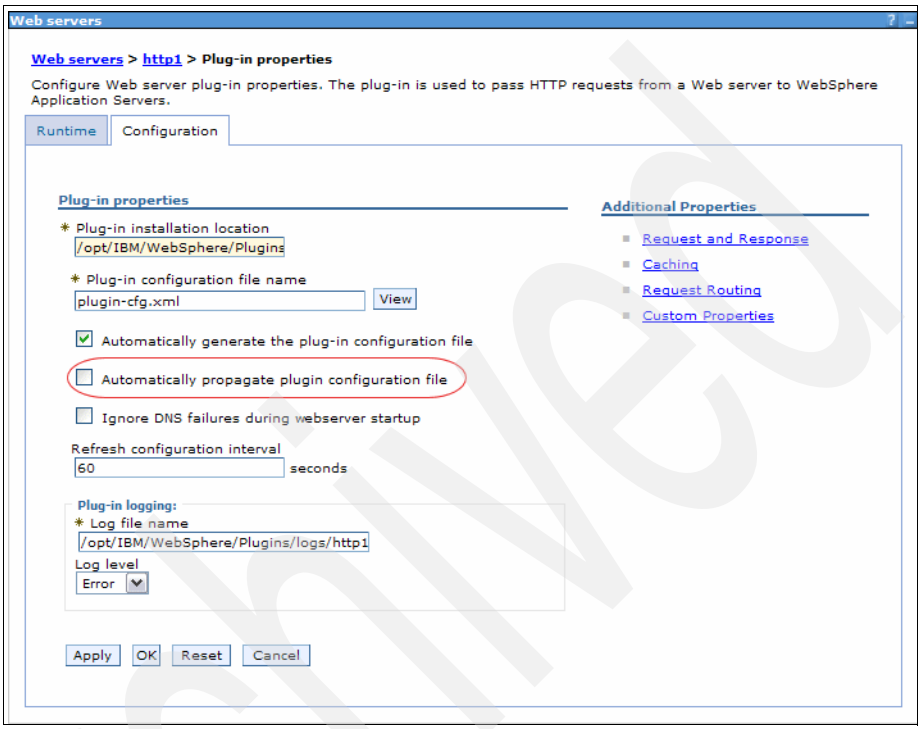


Figure 5-17 Disable automatic propagation of plug-in configuration file

3. On each HTTP server machine, copy the Web server plug-in file (plugin-cfg.xml) so you have a backup file for later restoration. Then edit the original plugin-cfg.xml file. Comment out the lines for the application servers of the other branch(es) so the HTTP server only knows about the application servers of its own branch.

   Note: You cannot use the Administrative Console to change this mapping as the console only allows you to change the mapping for the application itself, which means for the entire cluster and not for individual cluster members.

   Wait as long as the time specified in the RefreshInterval (at the top of the plug-in file), by default this is 60 seconds. After this time, the HTTP server reloads the plug-in file and your changes take effect.

The highlighted lines in Example 5-2 show which lines need to be commented out in the plugin-cfg.xml file. (In our example, we keep wasmember06 in the branch and remove wasmember05.)

Example 5-2 Isolate the branch by removing the server from the list of servers

```
...
<ServerCluster CloneSeparatorChange="false" LoadBalance="Round Robin"
Name="wascluster02" PostSizeLimit="-1" RemoveSpecialHeaders="true"
RetryInterval="60">
      <Server CloneID="10cp2mdfn" ConnectTimeout="0" ExtendedHandshake="false"
LoadBalanceWeight="2" MaxConnections="-1" Name="wasna02_wasmember06"
ServerIOTimeout="0" WaitForContinue="false">
         <Transport Hostname="wasmember06.ibmredbook.com" Port="9083"
Protocol="http"/>
         <Transport Hostname="wasmember06.ibmredbook.com" Port="9446"
Protocol="https">
            <Property Name="keyring"
Value="/opt/IBM/WebSphere/Plugins/etc/plugin-key.kdb"/>
            <Property Name="stashfile"
Value="/opt/IBM/WebSphere/Plugins/etc/plugin-key.sth"/>
         </Transport>
<!-- </Server>
      <Server CloneID="10cuccd8v" ConnectTimeout="0" ExtendedHandshake="false"
LoadBalanceWeight="2" MaxConnections="-1" Name="wasna01_wasmember05"
ServerIOTimeout="0" WaitForContinue="false">
         <Transport Hostname="wasmember05.ibmredbook.com" Port="9084"
Protocol="http"/>
         <Transport Hostname="wasmember05.ibmredbook.com" Port="9447"
Protocol="https">
            <Property Name="keyring"
Value="/opt/IBM/WebSphere/Plugins/etc/plugin-key.kdb"/>
            <Property Name="stashfile"
Value="/opt/IBM/WebSphere/Plugins/etc/plugin-key.sth"/>
         </Transport>
      </Server>
-->
      <PrimaryServers>
         <Server Name="wasna02_wasmember06"/>
<!-- <Server Name="wasna01_wasmember05"/> -->
      </PrimaryServers>
   </ServerCluster>
...
```

4. Quiesce the HTTP server at the Load Balancer and wait for a reasonable amount of time so all affinity requests to the HTTP servers have finished or expired, then stop the HTTP server.

For the Load Balancer, you can use the **dscontrol manager quiesce** command. The quiesce subcommand allows existing connections to complete (without being severed) and forwards only subsequent new connections from the client to the quiesced server if the connection is designated as sticky and stickytime has not expired. The quiesce subcommand disallows any other new connections to the server. You can check the number of connections to that server with the **dscontrol server rep ::** command. When there are no more active connections, you can safely shut the server down. See the Load Balancer Administration Guide Version 6.0 for more information about these commands.

5. Update the application and save it to the master configuration. Be careful *not* to synchronize with the nodes during this task.

   At this point the updated application resides in the master repository on the Deployment Manager system. The nodes and thus cluster members are not aware of the new code version and all active servers continue to serve the old application version that is stored in their local configuration repository subset.

6. Update the static content on the HTTP server.

7. Now you can synchronize the updated application code to the nodes of your first branch. Using the Administrative Console, go to **System administration** → **Nodes**. Select the appropriate node and click **Full Resynchronize**. See Figure 5-15 on page 167.

   The synchronized node can now serve the new application version while the other node or nodes of the other branch or branches continue to serve the old application version.

   If there is more than one node in your first branch, synchronize the other nodes accordingly. As a precaution you can test your application on the updated node before synchronizing the other node or nodes.

   Figure 5-18 shows the status during the update.

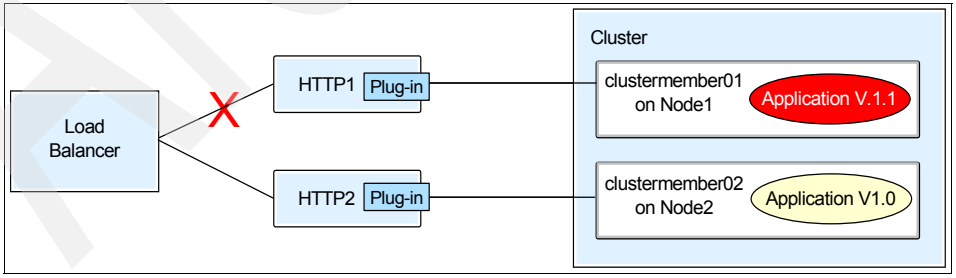


Figure 5-18 During the update

8. After testing it, make the updated branch accessible again:

   a. Start the application server cluster.

   b. Start the HTTP server or servers.

   Requests should now flow from the Load Balancer to all HTTP servers again. However, all requests being sent to the HTTP server of the first branch will use the new application version.

9. The next step is to update the second branch. Repeat steps 1 on page 168 to 8 for the second branch.

10. Restore the original plugin-cfg.xml files of the HTTP servers or remove the comment indicators from the files. This makes sure that all HTTP servers can send requests to all application servers again and that workload management and failover is available again.

11. Re-enable the file synchronization service for the Node Agents. Follow the steps described in “Deactivating the automatic file synchronization service” on page 165 but this time check the two fields. Save your changes and do not forget to restart the Node Agents.

12. The last step is to re-enable automatic propagation of the plug-in file. Follow the steps described in step 2 on page 168 and select **Automatically propagate plugin configuration file**. Save the changes.

Part 3

WebSphere HAManager

6

WebSphere HAManager

IBM WebSphere Application Server Network Deployment V6 introduces a new feature called High Availability Manager (commonly called HAManager) that enhances the availability of WebSphere singleton services such as transaction or messaging services. It provides a peer recovery mechanism for in-flight transactions or messages among clustered WebSphere application servers.

HAManager leverages the latest storage technologies, such as IBM SAN FS (Storage Area Network File System) 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 High Availability Clustered Multi-Processing (HACMP) or Tivoli System Automation (TSA).

This chapter discusses various of these HA scenarios.

6.1 Introduction to the HAManager

High Availability Manager (HAManager) enhances the availability of singleton services in WebSphere. These singleton services include:

- Transaction service - Transaction log recovery
- Messaging service - Messaging engine restarting

The HAManager runs as a service within each WebSphere process (Deployment Manager, Node Agents, or application servers) that monitors the health of WebSphere singleton services. In the event of a server failure, the HAManager will failover any singleton service that was running on the failed server to a peer server. Examples of such a failover include the recovery of any in-flight transactions or restarting any messaging engines that were running on the failed server. As depicted in Figure 6-1, 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 would exist in the topology. Every member in a WebSphere cluster knows where singleton services are running.

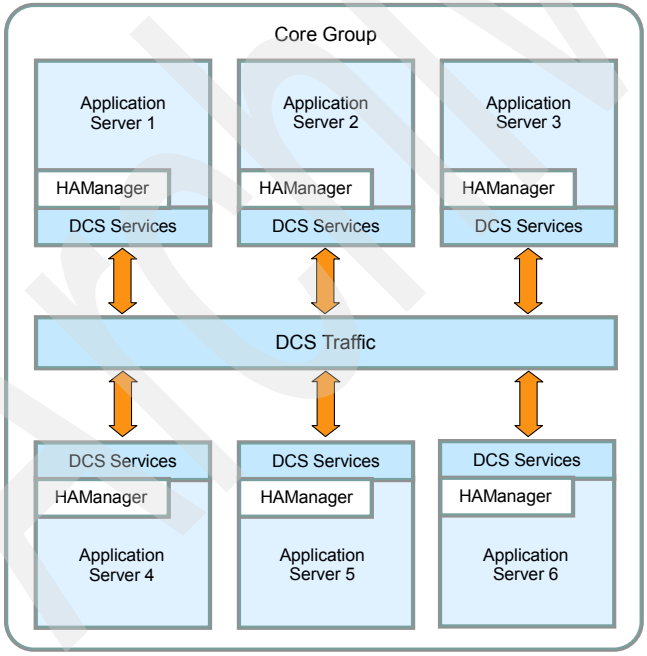


Figure 6-1 HAManager architecture

This peer-to-peer failover model dramatically improves recovery time. Also, 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 Part 5, “Using external clustering software” on page 283 for information about how to achieve this.

6.2 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 separate JVM. See Figure 6-2.

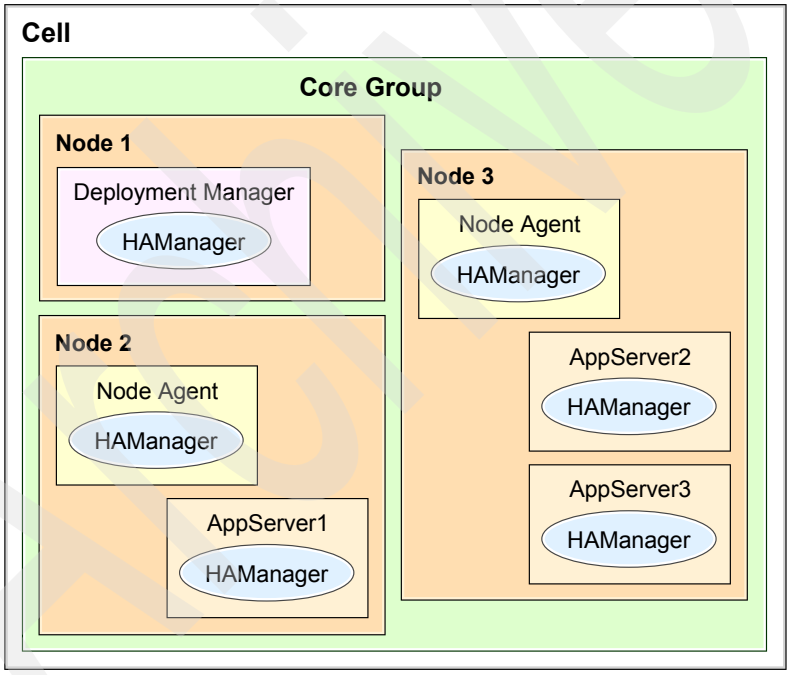


Figure 6-2 Conceptual diagram of a core group

A cell must have at least one core group. The WebSphere Application Server creates a default core group, called DefaultCoreGroup, for each cell. 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 6.2.4, “Core group policy” on page 188 and 6.3, “High availability group” on page 194.

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 scope of the singleton (such as a Transaction Manager or a messaging engine) is a WebSphere cluster then all members of the cluster are part of such a group of JVMs that can host the service.

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 share workload management routing information using the core group bridge service. See Figure 6-3.

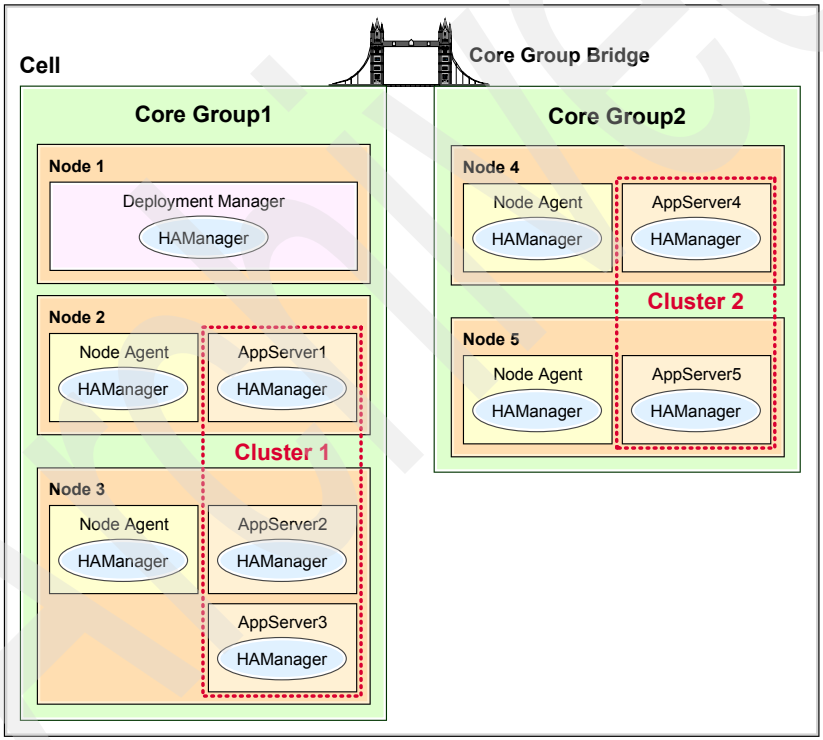


Figure 6-3 Conceptual diagram of multiple core groups

In a large-scale implementation with clusters spanning multiple geographies, you can create multiple core groups in the cell and link them together with the core group bridge to form flexible topologies. The most important thing is that every JVM in a core group must be able to open a connection to all other members of the core group.

Tip: Using core group bridges is a good way to handle intra-cell firewalls.

The core group bridge service can be used when configuring a backup cluster for EJB container failover. We have provided a step-by-step configuration example for the core group bridge service in 2.7, “Backup cluster support” on page 87.

6.2.1 Core group coordinator

After the membership of the core group stabilizes at runtime, certain members can be 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. See Figure 6-4 on page 180. 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.

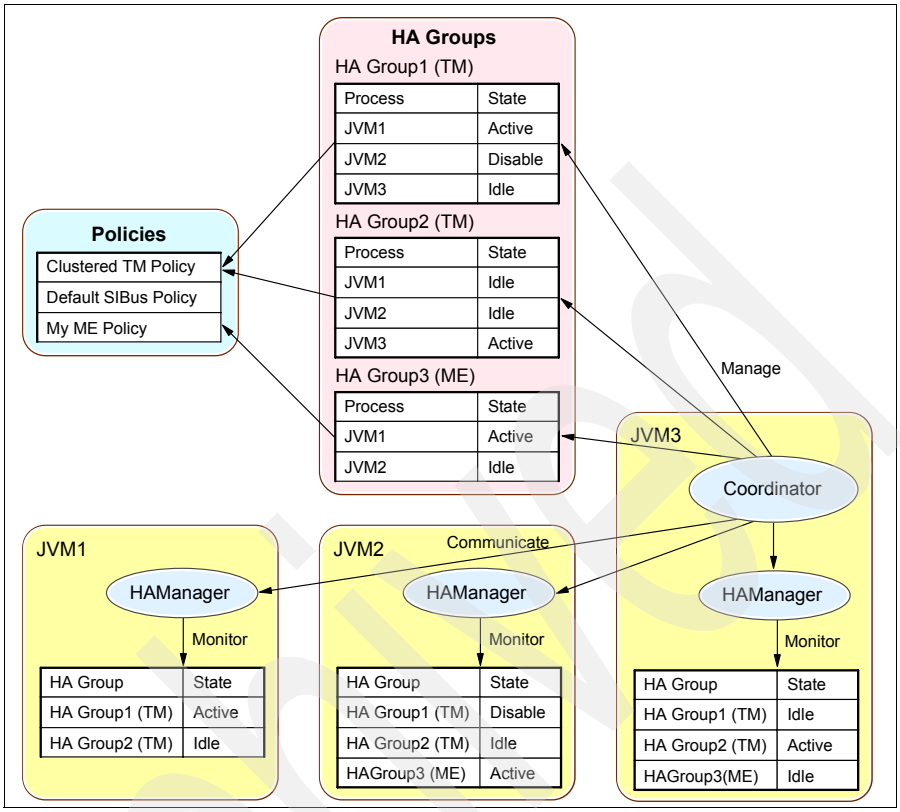


Figure 6-4 The responsibilities of a core group coordinator

Preferred coordinator servers

By default, the HAManager elects the lexically lowest named core group process to be the coordinator. The name of the process consists of cellname, nodename, and process name. For example in our environment, we have (among others), the following process names:

- wascell01/wasdgmr01/dmgr
- wascell01/wasna01/wasmember01
- wascell01/wasna01/nodeagent

In this example, the Deployment Manager has the lexically lowest name, followed by the Node Agent on wasna01, then wasmember01 and thus the Deployment Manager would become the coordinator server if no preferred coordinator has been configured. See Table 6-1 on page 183 for more examples of how the coordinator is elected (also when a preferred coordinator is specified).

Because a coordinator takes up additional resources in the JVM, you might want 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 processes. 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 were scheduled to run every two seconds, such a system would have the alarm firing all the time without being able to process the coordinator workload. 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 can 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 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 of 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.

Refer to the WebSphere V6 InfoCenter for more information about this topic.

To explain the election of coordinators, let us look at an example configuration with a core group consisting of the following JVMs in lexical order (all of them are in wascell01):

- wasdmgr01/dmgr
- wasna01/nodeagent
- wasna01/wasmember01
- wasna01/wasmember02
- wasna01/wasmember05
- wasna02/nodeagent
- wasna02/wasmember03
- wasna02/wasmember04
- wasna02/wasmember06

Table 6-1 lists the possible configurations of preferred coordinator servers and election results.

Table 6-1 Example configurations and results of preferred coordinator servers

Number of coordinators	Preferred coordinator servers	Inactive processes	Elected coordinators
1	nil	nil	wasdmgr01/dmgr
1	wasna01/wasmember01	wasna01/wasmember01	wasdmgr01/dmgr
2	wasna01/wasmember01	nil	wasna01/wasmember01 wasdmgr01/dmgr
2	wasna01/wasmember01 wasdmgr01/dmgr	nil	wasna01/wasmember01 wasdmgr01/dmgr
2	wasdmgr01/dmgr wasna01/wasmember01	nil	wasdmgr01/dmgr wasna01/wasmember01
2	wasna01/wasmember01 wasna01/wasmember02	wasna01/wasmember01	wasna01/wasmember02 wasdmgr01/dmgr
2	wasna01/wasmember01 wasna02/wasmember03	nil	wasna01/wasmember01 wasna02/wasmember03

Because 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 6-1.

Example 6-1 Message for a JVM becoming an active coordinator

```
[6/20/05 5:50:55:107 CDT] 00000018 CoordinatorIm I   HMGR0206I: The Coordinator
is an Active Coordinator for core group DefaultCoreGroup.
```

For JVMs that do not become coordinators, the message in Example 6-2 is displayed.

Example 6-2 Message for a JVM joining a view not as a coordinator

```
[6/20/05 5:50:55:112 CDT] 00000017 CoordinatorIm I   HMGR0228I: The Coordinator
is not an Active Coordinator for core group DefaultCoreGroup.
```

The preferred coordinator list can be changed dynamically. If there is a newly elected coordinator, a message as in Example 6-3 is written to the SystemOut.log.

Example 6-3 Message for an active coordinator retiring from the role

```
[6/20/05 5:54:58:994 CDT] 00000018 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 processor-intensive operation of a coordinator.

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.

These conditions are only a problem under the following circumstances:

- There are many 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 is not necessary to use more than a single coordinator. Normally, you use a preferred server to pin the coordinator to a server that does not start or stop typically. However, if that server fails, then the coordinator moves to the lexically lowest JVM.

6.2.2 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 might want to increase the buffer size if your WebSphere environment has high replication demands.

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.

When an application server is running low on the transport buffer, various messages are displayed in the SystemOut.log of the JVM as shown in Example 6-4.

Example 6-4 Congestion related messages in SystemOut.log

```
#### Messages indicating congestion ####
DCSV1051=DCSV1051W: DCS Stack {0} at Member {1}: Raised a high severity
congestion event for outgoing messages. Internal details are {2}.
...
DCSV1052=DCSV1052W: DCS Stack {0} at Member {1}: Raised a medium severity
congestion event for outgoing messages. Internal details are {2}.
DCSV
...
#### Message indicating that congestion cleared ####
DCSV1053=DCSV1053I: DCS Stack {0} at Member {1}: Outgoing messages congestion
state is back to normal.
```

These are not error messages. They are simply informational messages that indicate a congestion event has occurred in the transport buffer. The messages that are not sent during the congestion are retried later, or they are sent as a batch to use 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.

Setting the transport buffer size

The ideal setting for the transport buffer is very dependant on load but during some internal benchmarks, we used buffer sizes of 80 MB for a cluster that was processing 20 000 HTTP requests per second and each request resulted in 10 KB 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.

It is recommended to keep the transport buffer size setting symmetric on all processes. The amount of static storage that is consumed is actually quite small. For example, if you set the transport buffer size to 80 MB, only 12 MB actually gets statically allocated. The rest of the space is the limit of dynamic memory that is allowed to be allocated.

To change the transport buffer size, open the WebSphere Administrative Console, and click **Servers** → **Application servers** → ***<AppServer_Name>*** → **Core group service** (under Additional Properties). See Figure 6-5 for the configuration panel. Restart all processes in the core group after changing this setting.

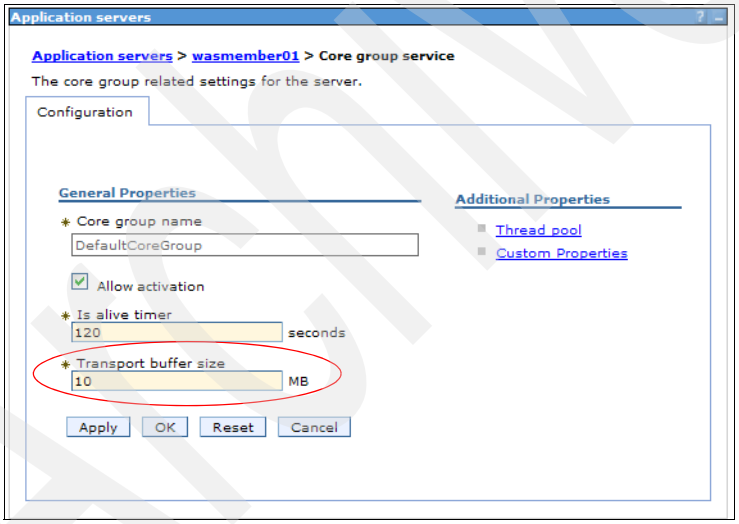


Figure 6-5 Change the transport buffer size of an application server

Note: This setting is per application server. You need to perform this change for all application servers in the core group.

Setting the IBM_CS_DATASTACK_MEG custom property

Whenever you change the transport buffer size, you should also increase the amount of heap space that in-flight messages are allowed to consume. This is changed by setting the IBM_CS_DATASTACK_MEG custom property for the core group. Its default value is 50.

We suggest to set this value to 100. Go to **Servers** → **Core groups** → **Core group settings** → **DefaultCoreGroup** (or click your core group) → **Custom properties** → **New**. Enter `IBM_CS_DATASTACK_MEG` for the Name and `100` into the Value field. See Figure 6-6. Restart all processes in the core group after changing this setting.

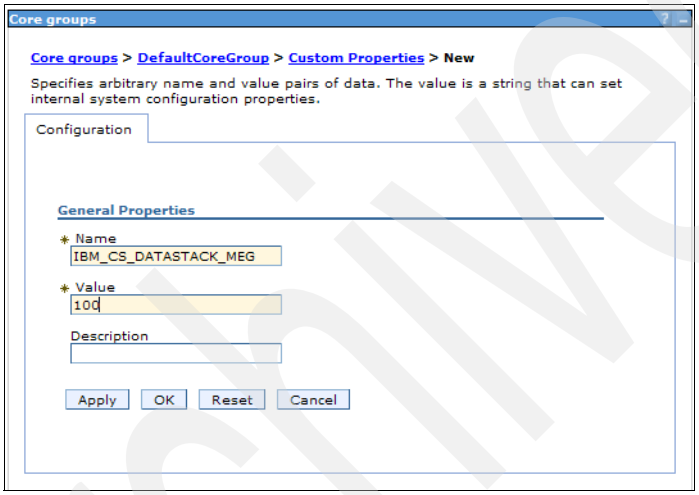


Figure 6-6 Core group custom property IBM_CS_DATASTACK_MEG

Refer to the WebSphere V6 InfoCenter for more information about this custom property.

6.2.3 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 healthiness 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.

6.2.4 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 refer to 6.3, “High availability group” on page 194. You can add, delete, or edit policies 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 show a core group policy, click **Servers** → **Core groups** → **Core group settings** → ***<core_group_name>*** → **Policies** → ***<existing_policy>*** (or **New**)**.** The panel in Figure 6-7 is shown.

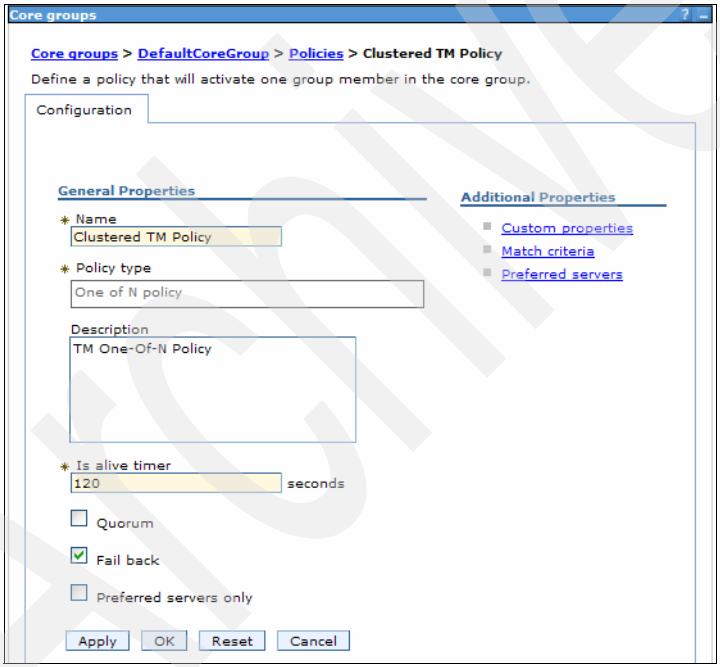


Figure 6-7 Editing or creating a core group policy

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

One of N policy

Only one server activates the singleton service at a time under this policy. If a failure occurs, the HAManager starts 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 highly available file system, such as Network Access Storage (NAS), that is available to all members. This is the recommended policy for systems that require automatic failover 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 option deselected.

  This option is only needed for WebSphere Extended Deployment customers using partitioning *and* using hardware to enforce quorums. However, if you are using WPF (Partition Facility) and the appropriate hardware, then contact IBM support to configure this setting correctly.

- Fail back

  When enabled, a singleton service is 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.

Attention: 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 IBM HACMP or Tivoli System Automation. 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 might be placed on a Journal 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 because all JVMs are running before a failover event. The expensive JVM start time is avoided during the critical failover time.

For more information about when and how to use this policy refer to Part 5, “Using external clustering software” on page 283.

Static policy

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 automatic failover is undesirable. If the server fails, the service can be moved to another server by updating the server name on the policy and saving it.

6.2.5 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 each of the available policies against the HA group name properties of the high availability group. The policy with the strongest match will be assigned to the HA group.

First the HAManager will find the set of policies that are eligible to govern the HA group. In order for a policy to be eligible, the match set must be a proper subset of the HA group name (all *<name,value> pairs* in the match set of an eligible policy must be in the name of the HA group). Secondly, the HAManager will choose the policy from the eligible list based on the number of matches. The eligible policy with the most name,value pair matches is selected.

Important: The results of these two steps must yield exactly one policy, otherwise the HAManager cannot assign a policy for the group.

Examples

In the following examples a name,value pair is represented using the notation name=value. Suppose you have the following policies:

1. Policy P1 has a match set of <name=Timmy>
2. Policy P2 has a match set of <name=Douglas>
3. Policy P3 has a match set of <age=5>
4. Policy P4 has a match set of <name=Timmy,age=5>

- When you have an HA group with the name of <name=Timmy>, policy P1 is the only eligible policy. P4 is not eligible because the match set for P4 contains a name,value pair of <age=5> that is not contained in the group name. Policy P1 is chosen for this HA group.
- Suppose you have an HA group with the name of <name=Timmy,surname=Stevens>. Again, policy P1 is the only eligible policy and will be chosen for this HA group.
- When you have an HA group with the name of <name=Douglas,age=5,surname=Berg>, policies P2 and P3 are both eligible because all the name,value pairs in the match sets of these policies are contained in the HA group name. However, both P2 and P3 each have one match. Neither one is better. The HAManager cannot choose a single policy, so neither is chosen (and an error message is e logged).
- Suppose you have an HA group with the name <name=Timmy,age=5,surname=Stevens>. Policies P1, P3 and P4 are all eligible because all the name,value pairs in the match sets of each of these policies are contained in the HA group name. P1 and P3 each have one match, while P4 has two matches. Therefore policy P4 will be chosen.
- When you have an HA group with the name <name=Harry> then there are no eligible policies. An error message will be logged.
- Suppose you have an HA group with the name <surname=Douglas>. There are no eligible policies (both name and value must match, thus <name=Douglas> and <surname=Douglas> do not match.) An error message will be logged.

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 6-2 and Table 6-3 for match criteria name, value pairs of messaging engines, and Transaction Managers, respectively.

Table 6-2 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 6-3 Match criteria for Transaction Managers

Name	Value	Match targets
type	WAS_TRANSACTIONS	All Transaction Managers
IBM_hc	Cluster name	All Transaction Managers in a cluster
GN_PS	Home server name	One particular Transaction Manager

6.2.6 Transport type

A transport type is the type of network communication a core group uses to communicate to its members. The following types of transports are 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.

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, 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.
- Unicast uses more ports than the Channel Framework transport.

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 20% 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 **`syncNode`** command line utility.

5. Start all Node Agents.
6. Start all cluster members and application servers.

6.3 High availability group

High availability groups are dynamic components created in a core group at run time. Each group represents a highly available singleton service. The available members in a group are ready to host the service at any time. See Figure 6-8.

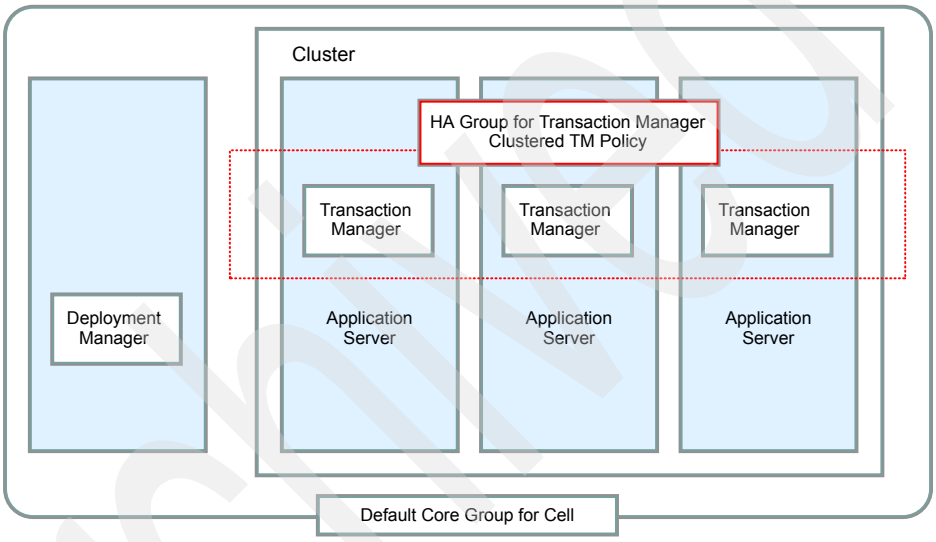


Figure 6-8 High availability group for a Transaction Manager

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 groups 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 6-9 on page 195.

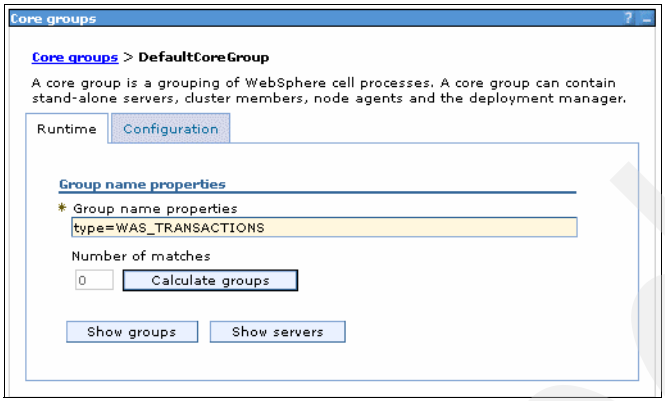


Figure 6-9 Find high availability groups for Transaction Managers

When you click **Show groups**, a list of high availability groups that match the criteria you specified is displayed as shown in Figure 6-10. Each high availability group is displayed along with its associated policy.

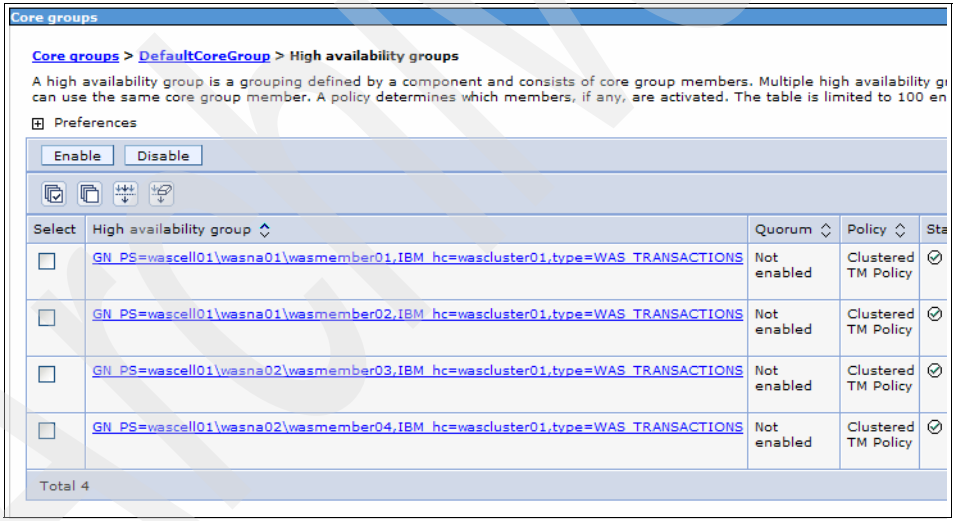


Figure 6-10 Listing Transaction Managers high availability groups

Select any high availability group and a list of group members is displayed as shown in Figure 6-11 on page 196. 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.

Note: Be aware that even though you can perform these actions on the Administrative Console, you cannot do something that conflicts with the core group policies. For example, suppose you had a One of N policy with a preferred server managing a messaging engine and the ME is running on the preferred server. If you try to move the ME using the Administrative Console, the policy overrides that and leaves the ME where it is running. In that case, the only choice you have is to change the policy.

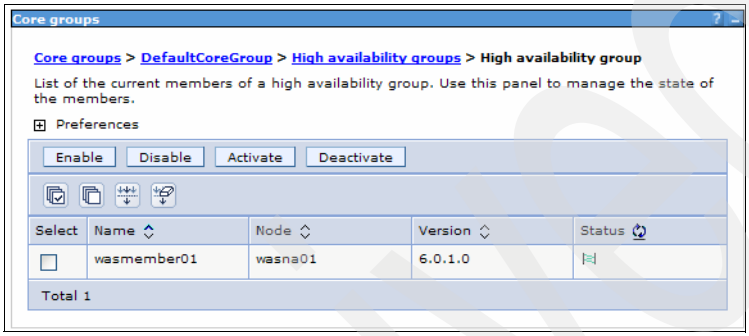


Figure 6-11 Managing a high availability group

6.3.1 State change of high availability group members

Here is an example for the one of N policy: There are two servers (A and B) in a group, only server A is a preferred server. HAManager activates the services (Transaction Manager or messaging engine) in server A, right after it joins the group. Later, server B starts and joins the group. Server B stays in idle state until server A fails. At that time, HAManager activates the services in server B until server A restarts. If the failback flag in the policy is set to true, when server A starts, HAManager deactivates the services in server B and activates them in server A again. If the failback flag is set to false, HAManager leaves server B as the active member.

Figure 6-12 shows the state changes for the group members.

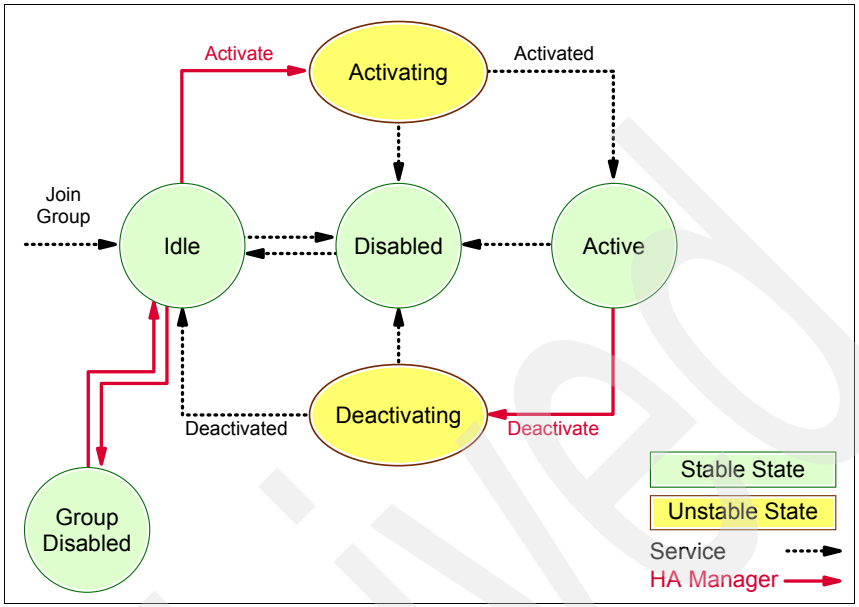


Figure 6-12 High availability group life cycle

6.4 Discovery of core group members

A JVM that is starting in a core group goes through the following 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 considers 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 6-5 Message HMGR0218I for a new view being installed

```
[6/20/05 7:28:44:458 CDT] 00000018 CoordinatorIm I   HMGR0218I: A new core
group view has been installed. The view identifier is
(9:0.wascell01\wasna01\nodeagent). The number of members in the new view is 6.
```

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 operating system might need to be tuned to prevent running out of ephemeral ports. Consult your operating system documentation for details.

6.5 Failure Detection

The HAManager monitors JVMs of core group members and updates the view accordingly. Changes in the view initiate failover, when necessary. It uses the following methods to detect a process failure:

- Active failure detection
- TCP KEEP_ALIVE
- Sockets closing

6.5.1 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 to 30 seconds is normally recommended for a well tuned cell.

Notes:

- For WebSphere V6.0.2, the default heartbeat settings have been changed to sending a heartbeat every 30 seconds and six consecutive lost heartbeats denote a failure.
- Settings as low as 6 seconds are possible but typically only used with WebSphere Extended Deployment WPF applications. Core groups requiring such a low setting should be kept small for best performance.

  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 6-6 on page 212. 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 starts. 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 runable threads accumulated in a system, each thread will suffer a long delay before getting scheduled. Threads of a JVM might 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 6-4 on page 200 to change the heartbeat frequency.

Table 6-4 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.

6.5.2 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 processor or 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 might have side-effects to other processes running in your system.

6.6 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, refer to Chapter 12 of *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392.

6.7 Transaction Manager high availability

The WebSphere Application Server Transaction Manager writes to its transaction recovery logs when it handles global transactions that involve two or more resources. Transaction recovery logs are stored on disk and are used for recovering in-flight transactions from system crashes or process failures. To enable WebSphere application server transaction peer recovery, it is necessary to place the recovery logs on a highly available file system, such as IBM SAN FS or NAS, for all the application servers within the same cluster to access. All application servers must be able to read from and write to the logs.

For a peer server to recover in-flight transactions, any database locks associated with the failed transactions should be released prior to the recovery. You need to use the lease-based exclusive locking protocol, such as Common Internet File System (CIFS) or Network File System (NFS) Version4, to access remote recovery logs from WebSphere application server nodes. Without the lease-based locking support, if one of the nodes crashes, locks held by all the processes on that node will not automatically be released. As a result, the transactions cannot be completed, and database access can be impaired due to the unreleased locks.

As depicted in Figure 6-13 on page 202, as two application servers perform two-phase commit (2PC) transactions, they place table or row locks in the database, depending on the configuration of lock granularity.

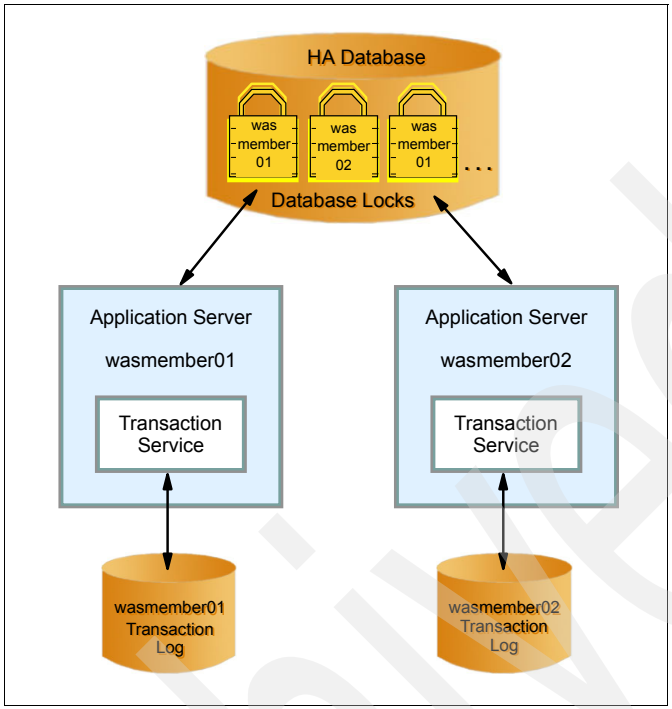


Figure 6-13 Two phase commit transactions and database locking

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 might leave locks in the database, which blocks the peer server from gaining access to the locked records.

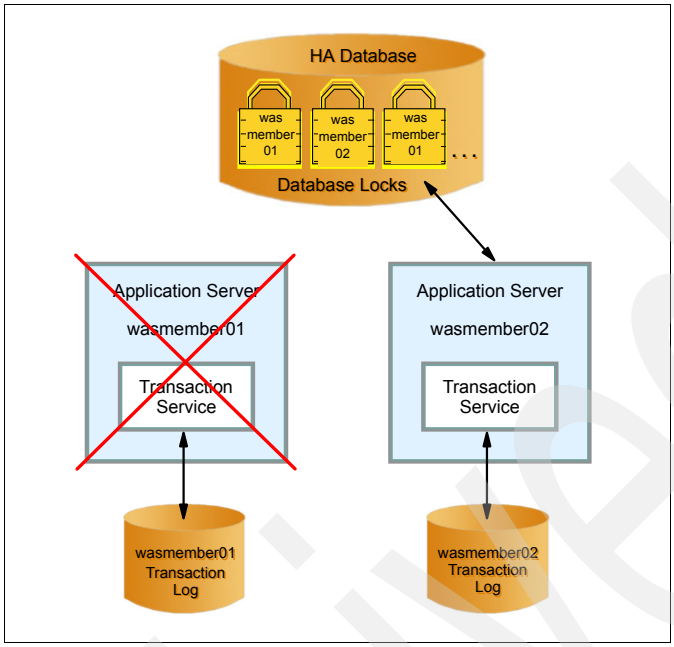


Figure 6-14 Server failure during an in-flight transaction

There are only two ways to complete the transactions and release the locks. One is to restart the failed server and the other is to start an application server process on another box that has access to the transaction logs.

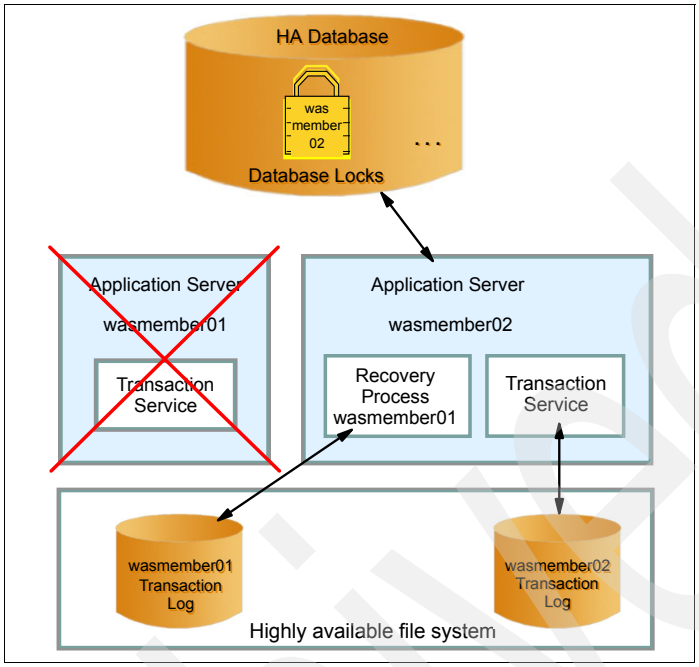


Figure 6-15 Recovery of failed transactions

Using the new HAManager support, a highly available file system and a lease-based locking protocol, a recovery process will be started in a peer member of the cluster. The recovery locks are released and in-flight transactions are committed.

We describe this configuration in detail in the next sections.

6.7.1 Transaction Manager HA of previous versions of WebSphere

In previous versions of WebSphere Application Server, transaction log recovery could only be achieved by restarting the application server which leads to slow recovery time from a failure (you can either restart the application server itself — automatically or manually — or use clustering software to failover to a backup server).

Starting a backup server using clustering software is known as a cold failover as the backup server needs to start an application server process during the failover. WebSphere also requires IP failover for transaction log recovery in older versions of WebSphere. Figure 6-16 on page 205 depicts a typical HA setup with previous versions of WebSphere Application Server:

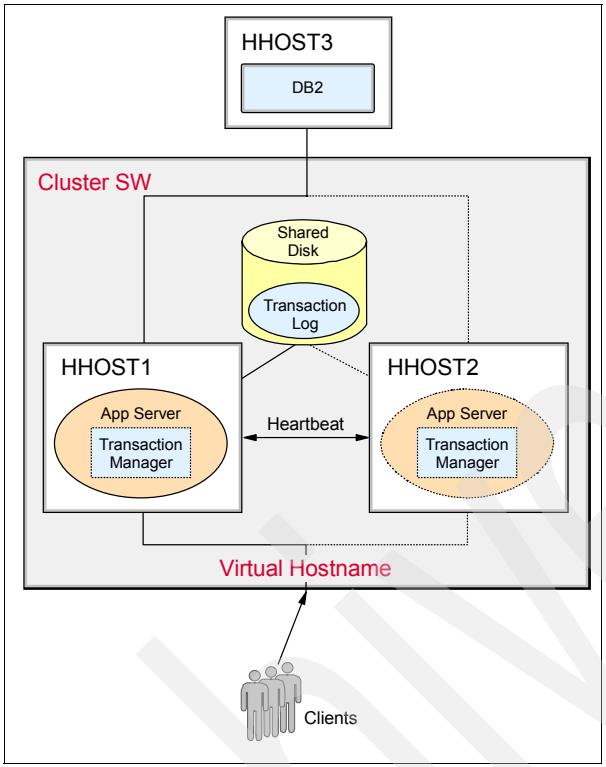


Figure 6-16 Traditional Transaction Manager high availability setup

A shared drive is attached to both servers. 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 or to 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.

By leveraging the 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 with external HA software, IBM WebSphere Application Server Network Deployment V6 itself can be a Transaction Manager HA solution with the right environment as described in the following section.

6.7.2 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 FS, where the transaction logs are stored, see Figure 6-17 on page 207.

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 backend database and then releases the transaction logs. No new work is performed beyond recovery processing. The Transaction Manager fails back when the failed server is restarted.

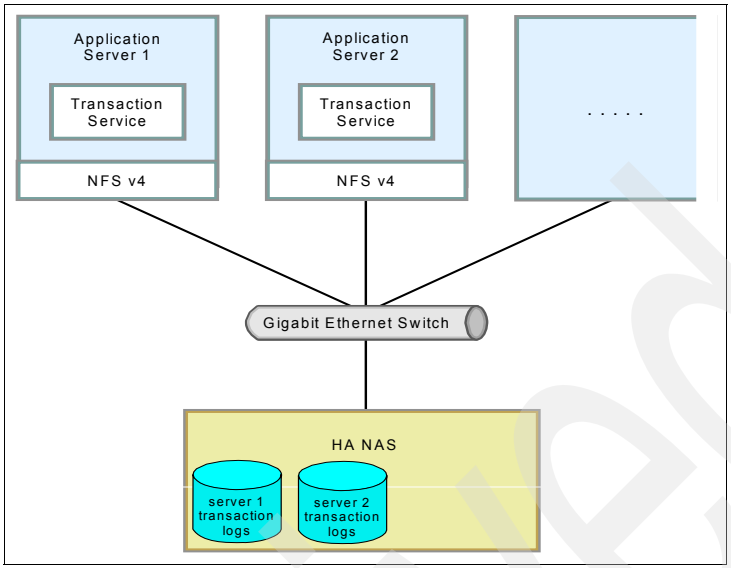


Figure 6-17 Network Deployment V6 transaction service high availability using NAS

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 when held by the failed member.

Important: All shared file systems that are compatible with WebSphere V6.0 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 failover in 10 seconds, this does not help if the lock lease time is 45 seconds, because the locks will not free up until 45 seconds. We recommend that you set the lock lease times to 10 seconds and the HAManager failure detection time to just over the lock lease time, 12 seconds (2 second for 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

Note: AIX 5.2 supports NFS version 3 while AIX 5.3 supports NFS version 4. For information about NFS V4, refer to *Securing NFS in AIX An Introduction to NFS v4 in AIX 5L Version 5.3*, SG24-7204.

However, basically any shared file system 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.

If you are unsure whether your selected file system supports this, see 6.7.4, "File System Locking Protocol Test" on page 213 for information about how to obtain a test program to verify this.

Tip: The white paper *Transactional high availability and deployment considerations in WebSphere Application Server V6* on IBM WebSphere Developer Technical Journal is an excellent description of Transaction Manager recovery in different environments. You can find this paper at:

`http://www.ibm.com/developerworks/websphere/techjournal/0504_beaven/0504_beaven.html`

For more information about IBM NAS and SAN FS technologies, go to the 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).

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 **Enable high availability for persistent services** on the Configuration tab shown in Figure 6-18.

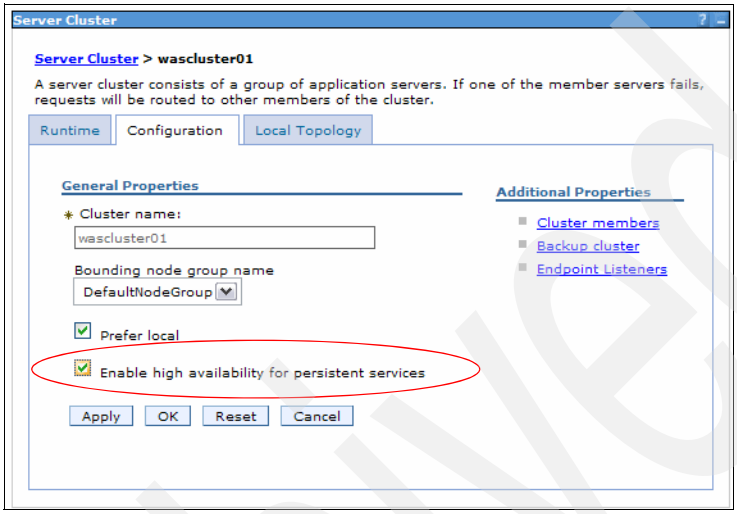


Figure 6-18 Enable high availability for the transaction service of a cluster

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. See Figure 6-19 on page 210.

Tips:

- It is recommended to 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 also work for WebSphere V6 transaction logs.

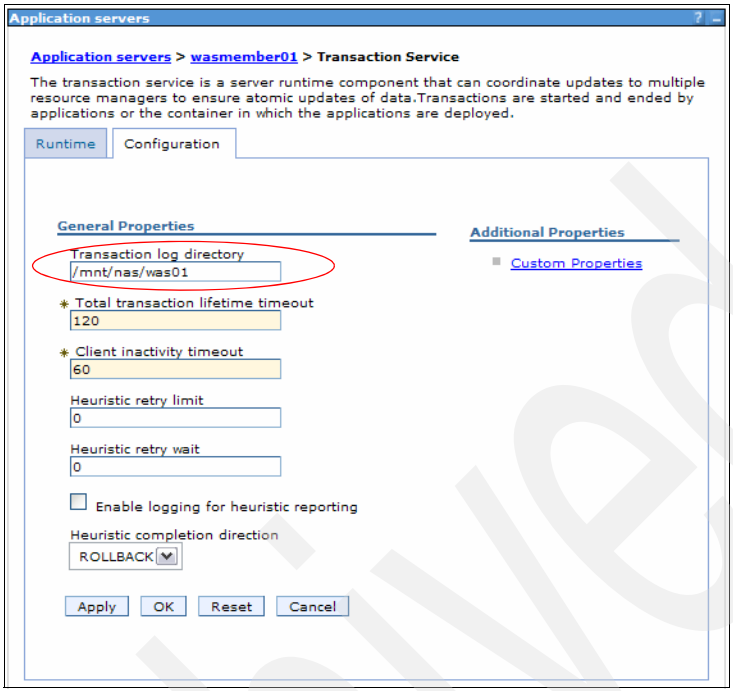


Figure 6-19 Change the transaction log directory of a cluster member

6. Save and synchronize the configuration.
7. Copy the existing transaction logs to the shared file system. Make sure the location and file permissions are correct.
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 6.3, “High availability group” on page 194.

Failover test

Now we perform a simple simulation of a JVM crash of application server wasmember01, and server wasmember02 performs the peer recovery of transaction logs for wasmember01. Application servers wasmember01 and wasmember02 run on different AIX nodes (wasna05 and wasna06 respectively). Application servers wasmember01 and wasmember02 are members of the same cluster, see Figure 6-20 on page 211.

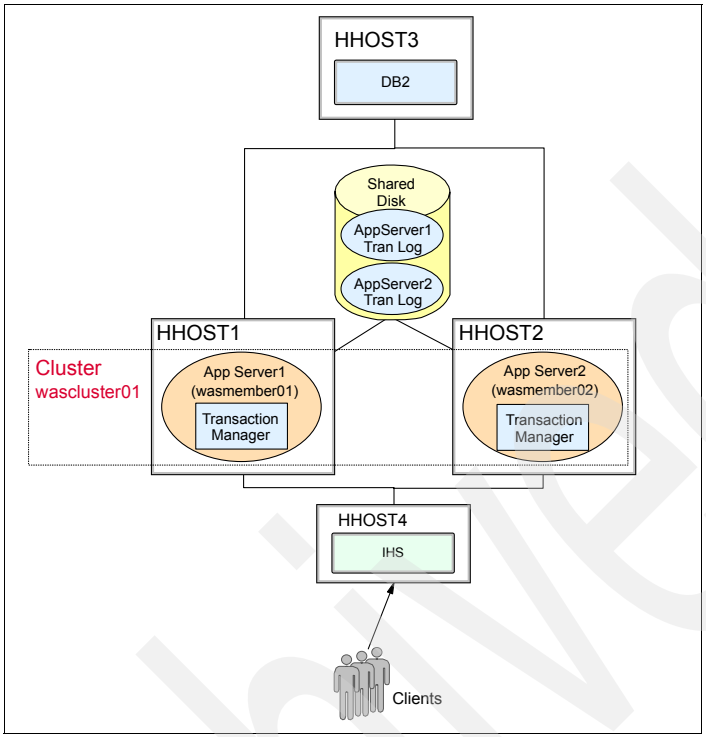


Figure 6-20 Test environment for transaction log failover

Important: It is important for you to 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.

However, 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 help you make the environment work.

Both application servers are up and running. We issue the AIX command **`kill -9 <pid_of_wasmember01>`** to terminate the process of wasmember01. The wasmember02 application server immediately detects the event from the closed socket connection and raises wasmember01 as a suspect. A new view is immediately installed to reflect the change. Then a recovery process is started in the wasmember02 JVM to recover the transaction logs of wasmember01. 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. See Example 6-6 on page 212 for the details.

Example 6-6 SystemOut.log of wasmember02 after terminating wasmember01

```
[5/2/05 9:57:21:598 CDT] 00000017 RmmPtpGroup   W   DCSV1113W: DCS Stack
DefaultCoreGroup at Member wascell07\wasna06\wasmember02: Suspected another
member because the outgoing connection to the other member was closed.
Suspected member is wascell07\wasna05\wasmember01. DCS logical channel is
View|Ptp.
[5/2/05 9:57:21:638 CDT] 00000017 DiscoveryRmmP W   DCSV1111W: DCS Stack
DefaultCoreGroup at Member wascell07\wasna06\wasmember02: Suspected another
member because the outgoing connection from the other member was closed.
Suspected members is wascell07\wasna05\wasmember01. DCS logical channel is
Connected|Ptp.
[5/2/05 9:57:21:657 CDT] 00000017 DiscoveryRmmP W   DCSV1113W: DCS Stack
DefaultCoreGroup at Member wascell07\wasna06\wasmember02: Suspected another
member because the outgoing connection to the other member was closed.
Suspected member is wascell07\wasna05\wasmember01. DCS logical channel is
Connected|Ptp.
[5/2/05 9:57:21:784 CDT] 00000017 VSync         I   DCSV2004I: DCS Stack
DefaultCoreGroup at Member wascell07\wasna06\wasmember02: The synchronization
procedure completed successfully. The View Identifier is
(75:0.wascell07\wasdmgr07\dmgr). The internal details are [0 0 0 0 0].
[5/2/05 9:57:21:989 CDT] 00000016 CoordinatorIm I   HMGR0228I: The Coordinator
is not an Active Coordinator for core group DefaultCoreGroup.
[5/2/05 9:57:22:013 CDT] 00000017 ViewReceiver  I   DCSV1033I: DCS Stack
DefaultCoreGroup at Member wascell07\wasna06\wasmember02: Confirmed all new
view members in view identifier (76:0.wascell07\wasdmgr07\dmgr). View channel
type is View|Ptp.
[5/2/05 9:57:22:013 CDT] 00000016 CoordinatorIm I   HMGR0218I: A new core group
view has been installed. The view identifier is
(76:0.wascell07\wasdmgr07\dmgr). The number of members in the new view is 4.
[5/2/05 9:57:22:187 CDT] 00000016 CoreGroupMemb I   DCSV8050I: DCS Stack
DefaultCoreGroup at Member wascell07\wasna06\wasmember02: New view installed,
identifier (76:0.wascell07\wasdmgr07\dmgr), view size is 4 (AV=4, CD=4, CN=4,
DF=5)
[5/2/05 9:57:22:801 CDT] 00000070 RecoveryDirec A   WTRN0100E: Performing
recovery processing for a peer WebSphere server (FileFailureScope:
wascell07\wasna05\wasmember01 [-572854721])
[5/2/05 9:57:22:894 CDT] 00000070 RecoveryDirec A   WTRN0100E: All persistant
services have been directed to perform recovery processing for a peer WebSphere
server (FileFailureScope: wascell07\wasna05\wasmember01 [-572854721])
[5/2/05 9:57:23:328 CDT] 00000070 RecoveryDirec A   WTRN0100E: All persistant
services have been directed to perform recovery processing for a peer WebSphere
server (FileFailureScope: wascell07\wasna05\wasmember01 [-572854721])
[5/2/05 9:57:23:818 CDT] 00000073 RecoveryManag A   WTRN0028I: Transaction
service recovering 2 transactions.
......
```

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.

6.7.3 Hot-failover of transaction logs using external HA software

If you have a specific requirement to use an external HA software in your environment, you can still take advantages of the hot-failover support available in WebSphere V6 or later.

You need to configure your clustering software in a way that when the active application server fails, the clustering software kicks off a script to activate the required services on the active peer server to take over the resource and continue. This feature provides a significant performance improvement by eliminating the server start time. For details on hot-failover of transaction logs using external HA software, refer to 9.6, “Transaction Manager failover with No Operation policy” on page 313.

6.7.4 File System Locking Protocol Test

The File System Locking Protocol Test indicates whether a shared file system supports the failover of transaction logs in WebSphere Application Server V6. The procedures and executable programs of this verification test determine if the shared file system supports the *lease based locking protocol*. Verification of the *write through to disk on flush* functionality of the shared file system cannot be reliably tested with a verification test because of the short timing window for testing this functionality.

The following Web site provides information about the File System Locking Protocol Test, including prerequisites and installation instructions, as well as the downloadable code:

`http://www.ibm.com/support/docview.wss?rs=180&uid=swg24010222`

Part 4

Platform specific information, IBM @server iSeries and zSeries

7

WebSphere HA on IBM @server iSeries

This chapter describes a recommended high-availability scenario for an application deployed to WebSphere Application Server Network Deployment V6 in an IBM @server iSeries environment.

WebSphere Network Deployment provides high availability for WebSphere-managed resources via WebSphere clustering and other Network Deployment options, which are described in various other chapters of this book, that are common to all WebSphere platforms. iSeries clusters provide the high availability for resources, such as database servers and file servers, that are outside the scope of WebSphere management.

Each of the technologies that we use is addressed individually at a high level for general understanding before going into detail about how to set up the iSeries cluster and make it usable by WebSphere. We then discuss the implications of using WebSphere Application Server in a highly available iSeries environment as well as configuration steps and considerations.

For our example we use independent ASPs and cross-site mirroring.

7.1 Introduction to iSeries HA

From its inception, the IBM @server iSeries system has been designed to run applications in support of core business processes. Features of the operating system and hardware have been designed to avoid unscheduled downtime whenever possible and to provide the ability to quickly restore the system to an operational state should a failure occur.

Today, iSeries systems are recognized as one of the most reliable and available servers in the marketplace. However, many companies are finding that even scheduled maintenance and the risk of unforeseen outages are unacceptable. This can be especially true in Web environments where you could have users or customers at any hour of the day. To reach this nearly continuous availability, eventually some type of clustering solution is required.

7.1.1 WebSphere Network Deployment: High availability for WebSphere processes

WebSphere Network Deployment should be used to configure horizontal clusters for enterprise applications; this enables high availability for WebSphere processes. Significant new options in WebSphere Network Deployment V6 include stateful session bean failover, peer recovery of WebSphere-managed transactions, and failover for the default messaging provider.

Refer to Chapter 2, “WebSphere Application Server failover and recovery” on page 35 and Chapter 6, “WebSphere HAManager” on page 175 for information about configuring and using these options.

7.1.2 iSeries clustering: High availability for other critical resources in the application path

Clustering in an iSeries environment is accomplished through Cluster Resource Services (CRS). Cluster Resource Services are a built-in part of the OS/400 operating system and, therefore, runs on each system in the cluster. CRS provides failover and switchover capabilities for systems used as database servers or application servers. CRS also provides the following functionality:

- Heartbeat monitoring for determining the state of systems in the cluster. If a failure occurs in one of the nodes, CRS alerts backup nodes to begin preparation for failover.
- Node communications allowing administrators on one system in the cluster to issue clustering commands to other machines (or nodes) in the cluster.

- A GUI interface makes it easier to set up and manage cluster implementation. Some examples of this functionality include:
  - Add a node to an existing cluster
  - Add a switchable hardware group to a cluster
  - Add a switchable software product to a cluster
  - Change the cluster description
  - Change the exit program name
  - Change the takeover IP address for a switchable software product
  - Delete a cluster
  - Start a cluster
  - Stop a cluster
  - Switch cluster resources from the primary node to the backup node
  - View messages relative to cluster activity
- Application recoverability via an API interface that maintains a program's state when switching between nodes in a cluster.
- Device resiliency, which allows control of switchable storage devices to be switched from a failing node in the cluster to a different node assigned as a backup.

One of the biggest benefits of clustering is the management of shared resources across the cluster. This includes managing a smooth transition of critical devices, data, and applications to a designated backup system in the event of a failure.

For the specific topology that we describe in this document, we use the device resiliency capabilities of iSeries clustering to enable high availability of the remote database servers as well as the file servers that host the WebSphere transaction logs.

7.1.3 Auxiliary Storage Pools (ASP)

iSeries single-level storage treats all storage as one large virtual address space (this includes main storage as well as disk storage). There is no concept of a disk volume or data set partition. However, the system provides the capability to separate this contiguous address space into smaller disk *pools* to make system backup and recovery faster and to provide Hierarchical Storage Management facilities. These pools are called *auxiliary storage pools*.

Conceptually, each ASP on the system is a separate pool of disk units for single-level storage. The system spreads data across the disk units within the ASP. If a disk failure occurs, you need to recover only the data in the ASP that contains the failed unit. The use of ASPs can reduce system backup time. To do this, create ASPs to include individual applications and data. A single ASP can then be backed up without impacting business operations while other applications that operate from different ASPs stay online.

An IASP can also be made dedicated, rather than switchable, for isolation of an application and its associated data.

7.1.4 Switchable disk pools (independent ASPs)

Independent auxiliary storage pools (IASPs) take the concept of ASPs further by making the ASP switchable between systems in a cluster when it resides on a switchable device. This could be an external expansion tower or an input/output processor (IOP) on the bus shared by logical partitions. The server that owns, or is attached to, the switchable device containing the independent disk pool, can then be switched either automatically, in the case of a failover, or manually by an administrator.

An IASP contains all of the information and data needed to make it an independently functioning storage device regardless of its controlling system. This includes having its own IFS which is mounted automatically as a UDFS onto the root directory for you when the resource is varied on. This means it also has standard IBM directories such as QIBM, including the ProdData and UserData directories. Although these resemble the ones in the system disk pool, they should not be confused. You can reference or change the current directory to /iaspName/ to view the IFS on the IASP.

It is also important to note that not all objects can be, or should be, installed into an IASP. Table 7-1 provides a list of object types supported by independent ASPs in OS/400 V5R3. Table 7-2 on page 221 lists object types that are not supported.

Table 7-1 Object types supported by IASPs in OS/400 V5R3

Object types supported by IASPs in V5R3				
ALRTBL	DTAQ	LIB	PAGSEG	SRVPGM
BLKSF	FCT	LOCALE	PDG	STMF
BNDDIR	FIFO	MEDDFN	PGM	SVRSTG
CHTFMT	FILE	MENU	PNLGRP	SYMLNK
CHRSF	FNTRSC	MGTCOL	PSFCFG	TBL
CLD	FNTTBL	MODULE	QMFORM	USRIDX
CLS	FORMDF	MSGF	QRYDFN	USRQ
CMD	FTR	MSGQ	SBSD	USRSPC
CRQD	GSS	NODGRP	SCHIDX	VLDL
CSI	IGCDCT	NODL	SPADCT	WSCST

Object types supported by IASPs in V5R3				
DIR	JOBD	OVL	SPLF	
DTAARA	JRN	OUTQ	SQLPKG	
DTADCT	JRNRCV	PAGDFN	SQLUDT	

Table 7-2 Object types not supported by IASPs in OS/400 V5R3

Object types not supported by IASPs in V5R3				
AUTL	DDIR	IGCSRT	MODD	PRDDFN
CFGL	DEVD	IGCTBL	M36	PRDLOD
CNNL	DOC	IMGCLG	M36CFG	RCT
COSD	DSTMF	IPXD	NTBD	SOCKET
CRG	EDTD	JOBQ	NWID	SSND
CSPMAP	EXITRG	JOBSCD	NWSD	S36
CSPTBL	FLR	LIND	PRDAVL	USRPRF
CTLD				

WebSphere installation

While the objects types might be compatible with IASPs, it is *not* recommended that you install licensed software or application code into an independent disk pool. Therefore, the WebSphere Application Server V6 binaries should be installed into the system disk pool. It is also recommended that your WebSphere V6 application code be deployed to the system disk pool. IBM WebSphere Application Server Network Deployment V6 provides the administrative functionality to deploy and promote your application code to all the nodes in your application server cluster.

However, the database and content for your application should be stored in the IASP and the application can then connect to its database in the independent disk pool at runtime. Alternatively, you can configure a connection pool to use the IASP, which is demonstrated in 7.2.6, “WebSphere data source configuration” on page 245). This provides the structure for a recovery environment for your applications in the event of a failover.

7.1.5 Cross-site mirroring

Another option that can be leveraged in a multisystem environment is cross-site mirroring or XSM. XSM provides a method of copying or duplicating the data in

an independent ASP to a second system which could optionally be geographically separated. We use cross-site mirroring in our sample scenario.

XSM uses a direct connection to the mirror system to write all changes to memory pages onto both systems in order to maintain data synchronization. When configuring cross-site mirroring you define logically equivalent independent ASPs on both systems. Then, when you start mirroring, XSM first copies all data from the designated primary copy to the mirror copy. When synchronized, each change to a page in memory on the primary will also be changed on the backup. Cluster Services can then provide failover support allowing you to switch to the backup system in a defined resource group in the event of a hardware failure or even environmental disaster.

Note: When mirroring is started, the IASP on the backup is in a *varied on* but not available state, which allows synchronization but does not allow use of or access to the data on the mirror copy. Use of the mirror copy requires a detach from mirroring and involves a full re-synchronization in order to resume.

Example

In the example shown in Figure 7-1 on page 223, the production site in New York has one IASP which is switchable between two systems, Node A and Node B. In a normal production environment, Node A is the primary system and has ownership of the IASP. In the event of a hardware failure, Node B can assume control of the IASP and continue operating as the production node in New York.

The IASP is also mirrored real-time to a second site in Boston, where there is a similar configuration. If a disaster occurs in New York, the mirror copy IASP in Boston would become the production copy and operations would resume on Node C or Node D. When the New York site is restored, the IASP there would resume as the mirror copy until synchronization is completed, at which time administrators have the option of keeping production in Boston or performing a controlled switch to bring production back to New York.

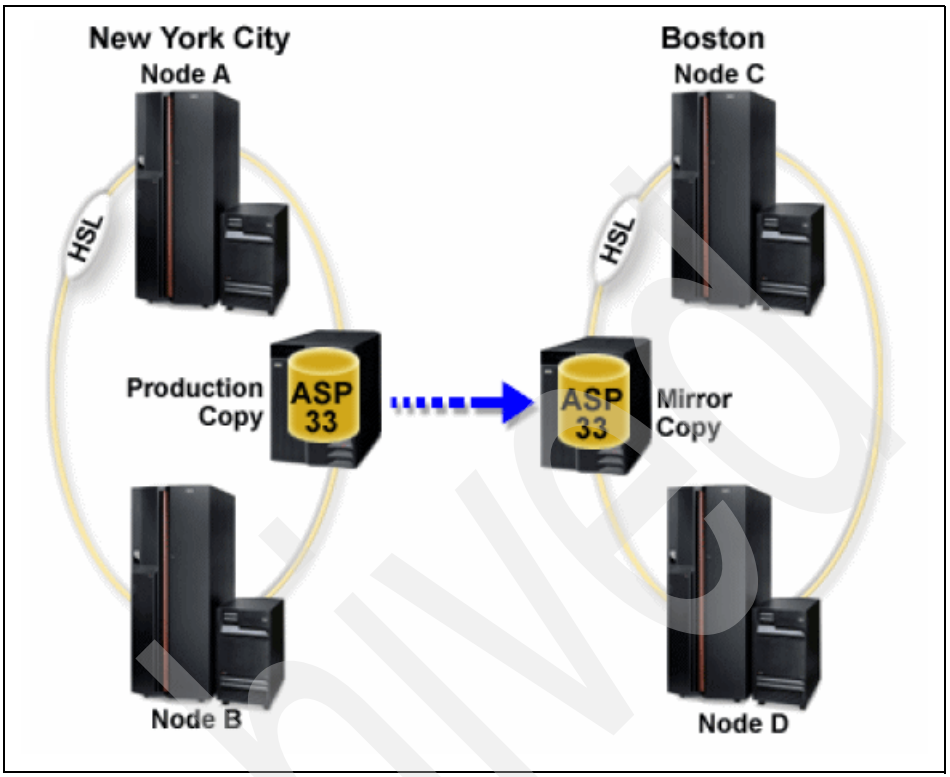


Figure 7-1 Double redundant solution using switchable ASP and geographic mirroring

7.1.6 Cluster resource groups

Resources that are available or known across multiple nodes within the cluster are called *cluster resources*. A cluster resource can conceptually be any physical or logical entity (database, file, application, device, and so forth). Examples of cluster resources include iSeries objects, IP addresses, applications, and physical resources.

Cluster nodes that are grouped together to provide resiliency for one or more clustered resources are called the *recovery domain* for that group of resources. A recovery domain can be any subset of the nodes in a cluster, and each cluster node might actually participate in multiple recovery domains. Resources that are grouped together for purposes of recovery action or accessibility across a recovery domain are known as a *Cluster Resource Group* or CRG. The CRG defines the recovery or accessibility characteristics and behavior for that group of resources. A CRG describes a recovery domain and supplies the name of the Cluster Resource Group exit program that manages cluster-related events for that group. One such event is moving the users from one node to another node

in case of a failure. There are three Cluster Resource Group object types that are used with Cluster Services:

- Application CRG

  An application resilient CRG enables an application (program) to be restarted on either the same node or a different node in the cluster.

- Data CRG

  A data resilient CRG enables data resiliency, so that multiple copies of data can be maintained on more than one node in a cluster.

- Device CRG

  A device resilient CRG enables a hardware resource to be switched between systems. The device CRG is represented by a configuration object with a device type of independent ASP. As of OS/400 V5R3, device CRGs are supported only for switchable hardware units, which include some disk resource or IASP.

With regard to the WebSphere application that we use in our example, only a device CRG is required to store the user databases and WebSphere transaction logs. Each CRG definition object can specify an exit program to be called for all actions on the cluster. The exit program is responsible for handling the action codes passed to it by the Cluster Resource Group Manager. Action codes are managed in the APIs that interact with the applicable CRG. The Cluster Resource Group exit program allows for customization in how the movement of the access point of a resilient resource takes place. Exit programs are written or provided by High Availability Business Partners and by cluster-aware application programs.

7.1.7 Device domains

A device domain is a subset of cluster nodes that share a set of resilient devices. A resilient device might be an independent ASP or any resource that is switchable between systems. The function of a device domain is to prevent conflicts that would cause an attempt to switch a resilient device between systems to fail. It is a logical construct within Cluster Resource Services. A device domain is used to ensure that, when a resource is given a name and identifying number on one system, all other systems in the domain reserve that number and name so that those configuration details are available on all nodes that might become the owning system in the event of a switchover or failover.

Resource assignments are negotiated with all other nodes of the domain at creation time to ensure that no conflicts exist and thereby ensuring uniqueness within the entire device domain. It is this construct that, even though only one

node can use a resilient device at any given time, ensures the device can be switched to another node and brought online.

The following are a few examples of resource identifiers that need to be unique in order to bring them online and are therefore negotiated across all systems in the device domain:

- IASP number assignments

  IASPs are automatically assigned a number to correlate the name of the IASP. The user chooses the resource name. The system manages the assigned IASP numbers, which might not be in numerical order. The order depends on a number of factors, including the creation date and the creation of IASPs on other nodes in the device domain.

- DASD unit number assignments

  To keep from conflicting with the permanently attached disk units of each node, all IASP unit numbers begin with a 4. IASP disk unit numbers start with the number 4001.

- Virtual address assignments

  The cluster configuration determines the virtual address space required for the IASP. Virtual address assignments (the cluster configuration) are ensured not to conflict across all nodes in the device domain.

7.2 Sample scenario configuration

Now that we have provided an overview of some of the clustering components used in iSeries clustering, we detail how to set up such an environment and discuss the implications for WebSphere V6 in iSeries environments. This involves creating an IASP and cluster, creating and adding nodes to the device domain, configuring cross-site mirroring and then how to set up your applications to utilize this highly available architecture.

We use the IBM Trade Performance Benchmark Sample for WebSphere Application Server (called Trade 6 throughout this book) as our WebSphere application. Trade 6 requires a special installation script in iSeries environments. We also assume that the database is on an iSeries system. Therefore, we do not recommend to use the Trade 6 version that can be downloaded from the internet. Instead, we provide the Trade_iSeries.zip file in the redbook repository. This zip-file contains the trade.ear file as well as the iSeries installation script and an OS/400 save file containing the database. Refer to Appendix B, “Additional material” on page 603 for download and installation instructions.

Figure 7-2 on page 226 shows our sample scenario.

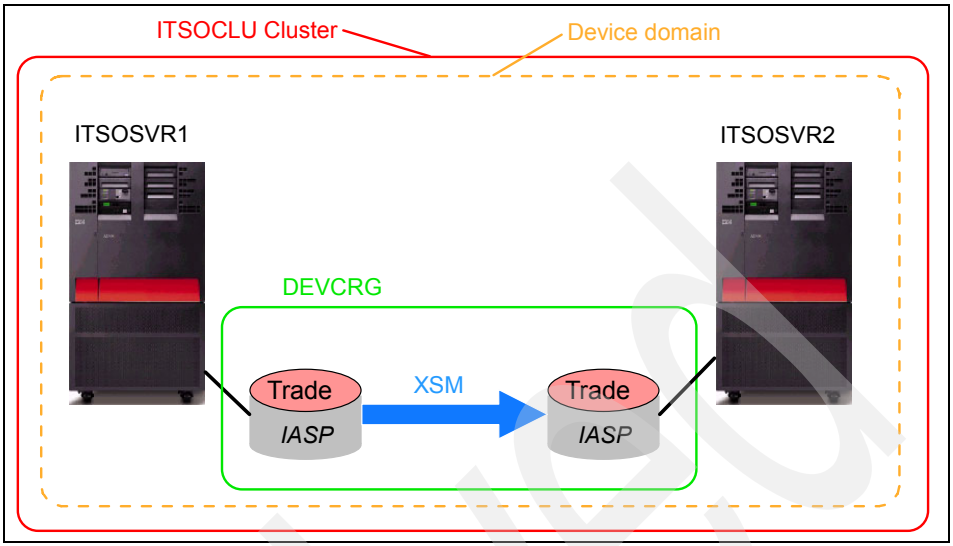


Figure 7-2 Redbook sample clustering and cross-site mirroring configuration

7.2.1 Create independent disk pool

The name of our IASP is trade. The two systems where the IASPs eventually reside are called ITSOSVR1 (primary) and ITSOSVR2 (backup). The Trade 6 database is installed on the IASP.

Important: We can create the IASP first, before creating the cluster, because we only create one IASP on a single node. If you need to create multiple IASPs on separate nodes, then you must create the cluster and assign the device domain first. Otherwise, you might get resource assignment conflicts.

Planning

There are several things to consider before creating your IASP:

- Determine how much space you need. Although you can add more disk units at a later time it is easiest if you consider future requirements up front.
- Only unallocated disk units can be allocated to an independent ASP. You, therefore, need to have space in your system for new disk units or you need to clear some existing disks. Remember to consider how this affects your existing RAID parity sets and how you will be configuring parity for the new IASP disk units.
- In our scenario, we implemented cross-site mirroring which requires both the production copy IASP and the mirror copy IASP to be nearly equal in size.

The best way to do this is to use disk units with the same capacities for both copies.

- For best performance of your WebSphere application, you must also consider the number of disk units or read/write arms in your disk configuration.
- If you intend to configure your IASP as a switchable tower between two systems, then you need to make sure that your devices are part of a switchable entity or tower. Remember that all devices associated with that tower will be switched during a failover!

Creating the IASP

Although it is possible to use the green screen interface, the simplest and recommended method to configure and manage an IASP is using the iSeries Navigator GUI. Access to disk functions is controlled via Service Tools user profiles. Make sure that you have access to a user profile authorized for SST before proceeding.

This section highlights the aspects of creating an IASP using the iSeries Navigator. You can find step-by-step instructions for creating IASPs in Chapter 7 of *Clustering and IASPs for Higher Availability on the IBM @server iSeries Server*, SG24-5194.

To create the IASP:

1. From iSeries Navigator under the menu for the primary system, click **Configuration and Service** → **Hardware** → **Disk Units**.
2. Enter the SST user ID and password into the Service Device Sign-on dialog box and click **OK**.
3. Under Disk Units right-click **Disk Pools** and select **New Disk Pool** from the context menu (see Figure 7-3 on page 228).

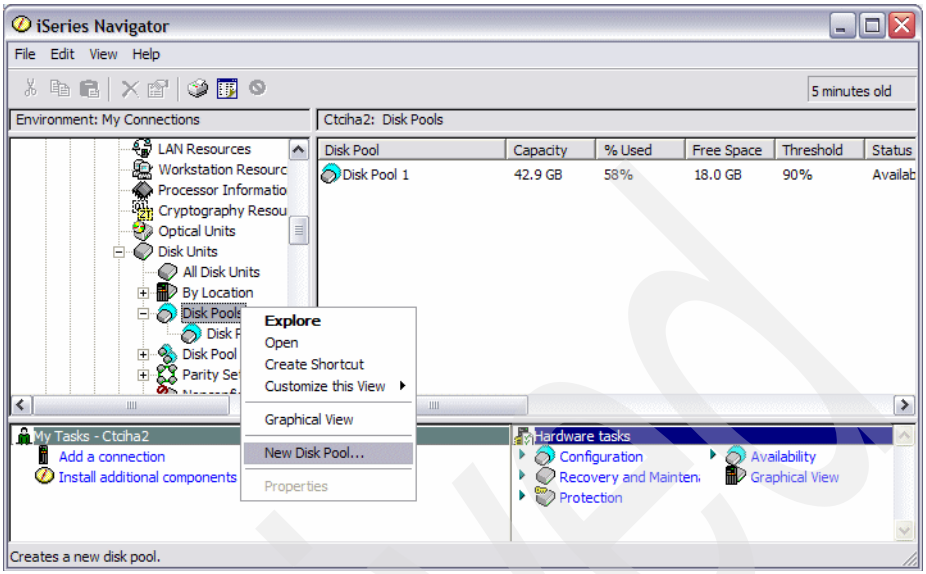


Figure 7-3 Create new disk pool with iSeries Navigator

4. This starts the New Disk Pool wizard with the Welcome page. Click **Next**.
5. The New Disk Pool panel appears next. Choose **Primary** for the type of disk pool and give the pool a name, we use trade for our example. The database field allows you to customize the name of the database that will be used by the system to create a relational database entry. Leaving this field as Generated by the system defaults the database name to the name of the disk pool (see Figure 7-4).

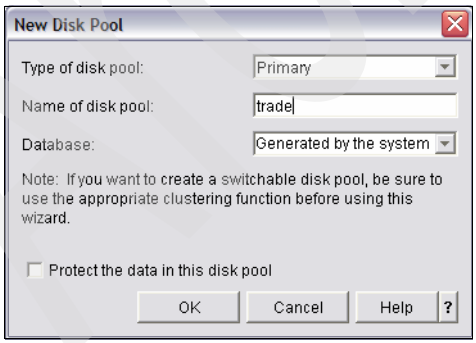


Figure 7-4 New Disk Pool panel

6. On the Select Disk Pool panel, select **Next**. This panel allows you to select the disk pools that you want to add disk resources to and should have already have your new IASP selected, as shown in Figure 7-5.

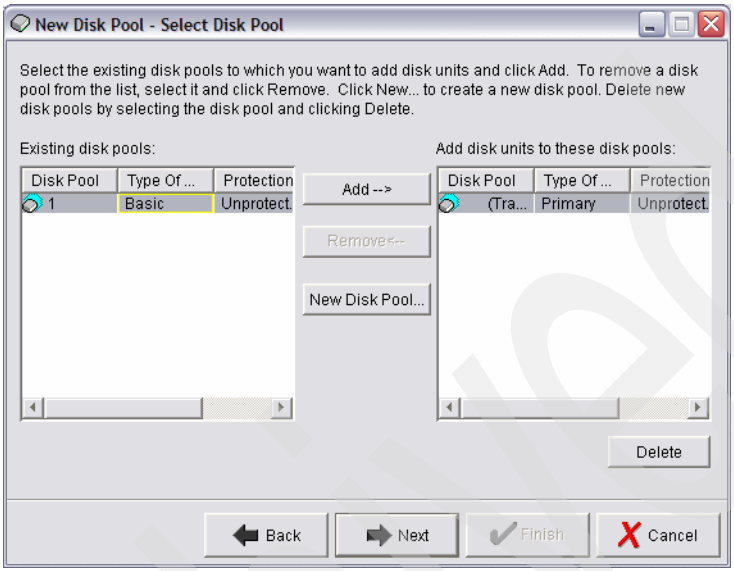


Figure 7-5 Select pools to which to add disk units

7. The Add Disk to Pool window is shown. Now that we have a Disk Pool defined, we need to add unallocated disk units to the pool. Click **Add Disks** to add disks to the pool.

> **Note:** If you selected the Protect the data in this disk pool box on the New Disk Pool panel, then you get two buttons called Add Disks to be Mirrored and Add Parity-Protected Disk instead of the Add Disks button.

8. The Add Disks panel allows you to select disk units to be added into the independent ASP. Remember that only unconfigured disk units are eligible. Therefore, before you start the configuration, it is important to have unallocated disks available for the disk pool. Because we do have an unallocated disk, it appears on this panel. Select the disk or disks that you would like to use, and click **Add** to add them to the pool (see Figure 7-6 on page 230).

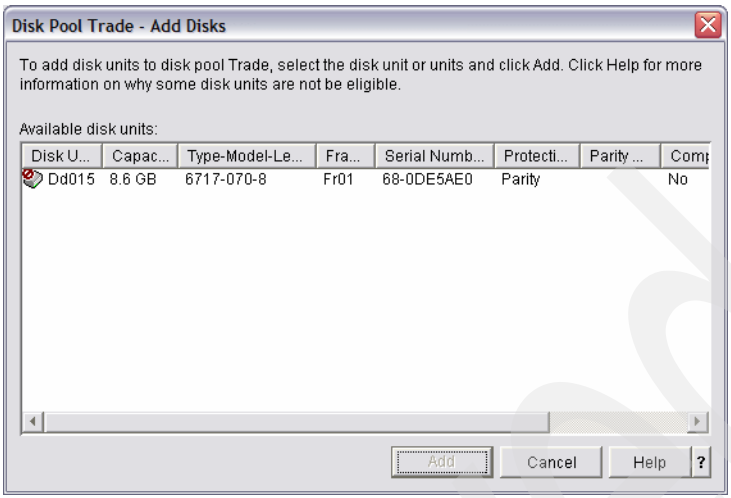


Figure 7-6 Add Disks to the new ASP

9. Verify the disk unit that were added to the disk pool. Make sure that the selection you made is correct, and select **Next** to continue.
10. Verify the disk pool configuration. Make sure that the disk pool configuration is correct, and click **Finish** to create the pool, as shown in Figure 7-7.

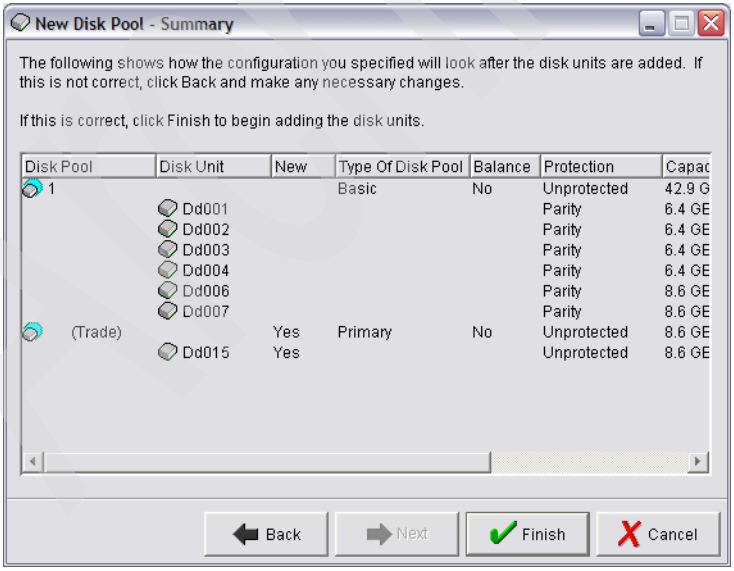


Figure 7-7 Final summary of disk pool before creation

After clicking Finish, a dialog box that shows the progress of the command is shown. The disk pool is now in the process of being created. There is also a level of formatting and preparing each of the disks in the IASP so this can take several minutes.

11. Before the command completes, you might see a message box warning you that the IASP is not associated with a device group. You can ignore this message for now and continue as we will be creating the clustering portion next. Select **Complete** to finish the creation wizard.

The wizard has completed and you should see the IASP named Trade in the Disk Pools list, as shown in Figure 7-8. The next step is to configure the cluster.

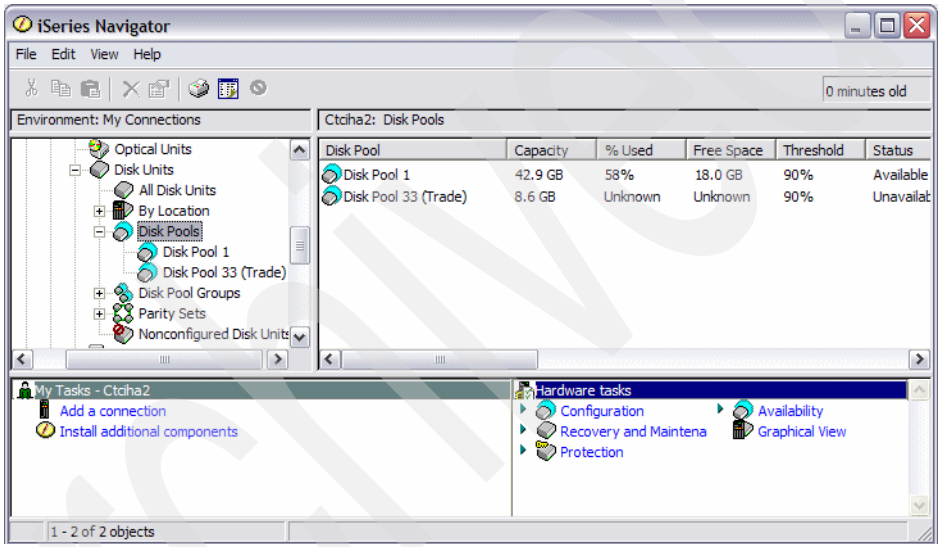


Figure 7-8 iSeries Navigator view of newly created disk pool

7.2.2 Configuring the cluster and resource group objects

The next step is to create the cluster. A cluster describes the nodes (machines) that participate in the cluster and the respective interface address. The cluster interface address is an IP address which is used by Cluster Resource Services to communicate with other nodes in the cluster.

To prepare for configuring clusters using the iSeries Navigator GUI, you should make sure that the iSeries environment is set up and configured as described in this section. Therefore, prior to creating a cluster, make sure that all requirements are met and necessary configuration tasks completed.

These are the steps for creating the clustering components:

1. Prepare your server by creating the necessary TCP/IP interfaces and setting the system attributes to allow clustering (see “Verifying or installing software and other system requirements” on page 232 and “TCP/IP requirements” on page 232).
2. Create the cluster and duplicate it on the second machine (see “Creating the cluster” on page 233).
3. Create a cluster resource group to associate the IASP into (see “Creating a cluster resource group” on page 237 and “Configuring the recovery domain and data ports” on page 238).

Verifying or installing software and other system requirements

You need to verify or install Product Option 41 - OS/400 - HA Switchable Resources (Licensed Program Product 5722-SS1) which is required for clustering and for sharing resources.

In addition, the follow system requirements must be met:

- The QUSER user profile must be enabled.
- The Allow add to cluster (ALWADDCLU) parameter of the Change Network Attributes (CHGNETA) command must be set to *ANY to allow other cluster nodes to add this node to their cluster.

TCP/IP requirements

TCP/IP requirements include the following:

- TCP/IP must be started on every node chosen to be in the cluster (STRTCP).
- TCP Server *INETD must be active on all nodes in the cluster (STRTCPSVR *INETD). Verify this by checking for the presence of a QTOGINTD (user QTCP) job in the Active Jobs list on the subject node. *INETD provides a port in the TCP connection list that listens for various clustering functions.
- Ports 5550 and 5551 are reserved for IBM clustering and must not be used by other applications. Use the **NETSTAT** command to review port usage. Port 5550 is opened by clustering and is in Listen state when *INETD is started.

> **Tip:** Configure *INETD to start automatically when TCP/IP is started. In iSeries Navigator, select ***System_name*** → **Network** → **Servers** → **TCP/IP**. Right-click **INETD** in the right-hand pane, select **Properties**, and enable **Start when TCP/IP is started**.

- The cluster needs several IP interfaces on the cluster nodes. These interfaces are:
  - The *takeover IP address* which is assigned to the CRG and is activated on the system that becomes the primary. We use the IP address 192.168.100.150 in our scenario. Configure this interface with the AUTOSTART parameter set to *NO.
  - At least one TCP/IP interface on each of the cluster nodes for the *cluster communication*. However, it is recommended to use multiple interfaces, preferably on separate network adapters. These interfaces are needed when creating the cluster (see "Creating the cluster" on page 233). For our scenario we use the following addresses:

Node	IP Address
ITSOSVR1	192.168.100.101
ITSOSVR2	192.168.100.102

  - At least one TCP/IP interface on each of the cluster nodes for the *cluster replication*. These IP addresses are used as the data port addresses when configuring the recovery domain (see "Configuring the recovery domain and data ports" on page 238).

    It is highly recommended to use a high-speed connection, for example OptiConnect, for the cluster replication. It is also recommended to have redundant data ports to ensure availability of the cluster replication in case of a communications device failure. In addition, having multiple data ports also improves the cluster replication performance.

Note: You can use the same physical device for the cluster communication interface and the takeover IP address.

- Configure the TCP loopback address of 127.0.0.1. It must show a status of Active. Use the **NETSTAT** command to verify the status on the subject node.

Creating the cluster

To create the cluster:

1. In iSeries Navigator, under Management Central, ensure that your central system is the system that you want to be your primary cluster node (with the IASP). If this system is not the current central system, change it by right-clicking and selecting **Change Central System.**
2. Right-click **Clusters** under the Management Central connection and select the option to create a **New Cluster** from the menu, as shown in Figure 7-9 on page 234.

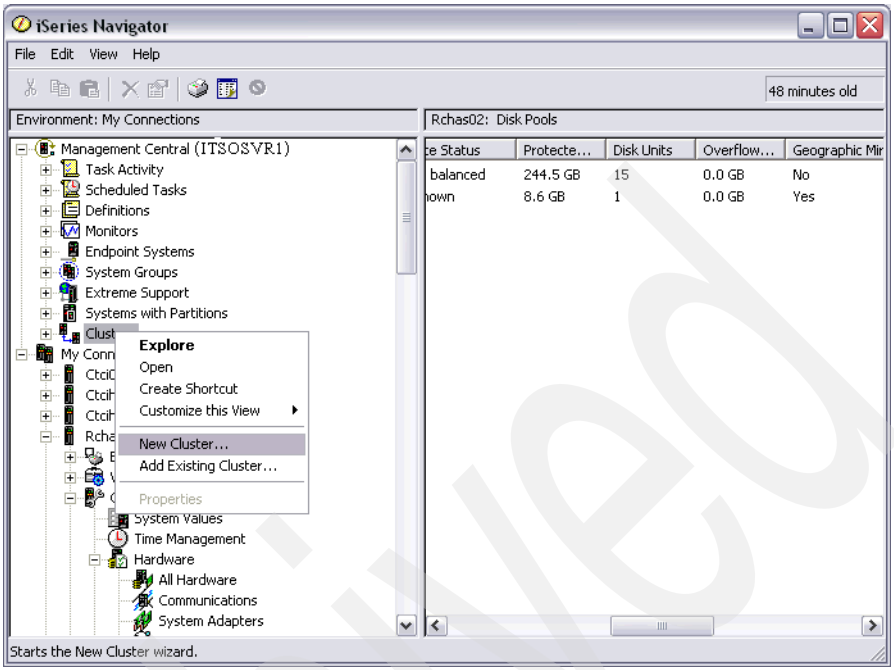


Figure 7-9 Start the new cluster wizard

3. Select **Next** on the New Cluster wizard.
4. Specify the cluster name and press **Next**. In our scenario, we use the name ITSOCLU.
5. The next panel asks you for details about the cluster member:
   a. Specify the name that you would like clustering to use as a reference for this system. This can be any name, but typically the system's host name is used. We use ITSOSVR1 for our configuration.
   b. Specify the server's host name in the second field. In our case, this is again ITSOSVR1.
   c. The next two fields are for the IP addresses that clustering uses for communication between the nodes in the cluster. These are the interfaces that you created in "TCP/IP requirements" on page 232 and which are used at this point. For higher availability you should use two separate lines and interfaces for cluster communications. However, we have only one interface available on each system so we can only specify one. For system ITSOSVR1, this value is 192.168.100.101, which we enter into the Cluster interface IP address 1 field.
   d. Click **Next** after you have filled in all values.

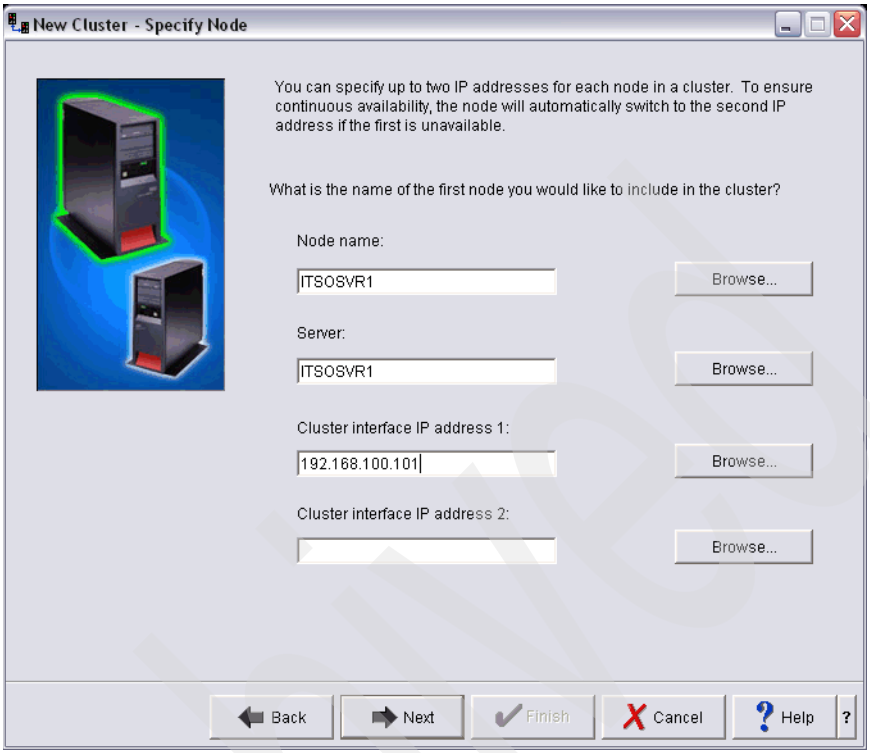


Figure 7-10 Specify cluster node

6. Specify the second node ITSOSVR2 and the IP interfaces for clustering communication and click **Next** again.
7. If a warning message is presented indicating that no switchable software is found, select **Next** to continue and create the cluster.
8. The cluster creation status is displayed on the Create Cluster window. Click **Finish** when the creation process has completed.

You have created a cluster. Figure 7-11 on page 236 depicts the current configuration.

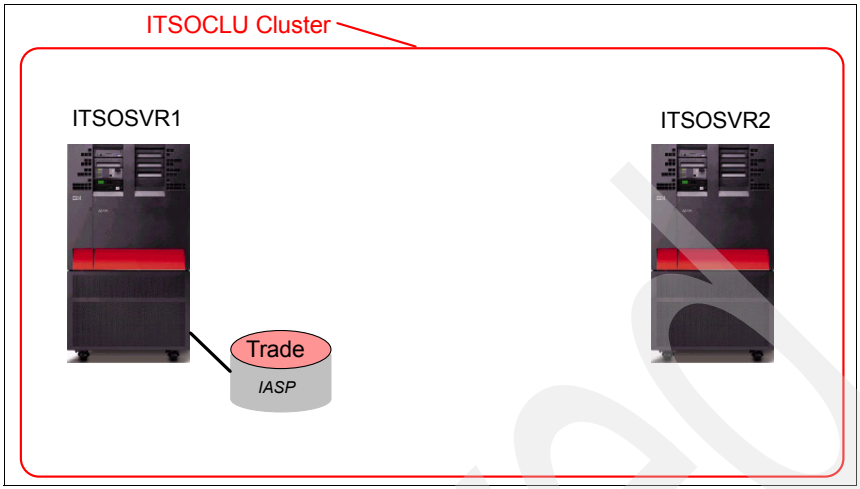


Figure 7-11 Sample cluster configuration - current status

Starting the cluster

At this point you should start the cluster on both nodes. You can do this by issuing the following commands:

```
STRCLUNOD ITSOCLU ITSOSVR1
STRCLUNOD ITSOCLU ITSOSVR2
```

You can also right-click each node in the cluster under **Management Central** → **Clusters**, and choose **Start**.

Adding device domain entries

When using iSeries Navigator to create a cluster, a device domain with the same name as the cluster is added to all nodes in the cluster automatically. However, you must still add the device domain entries. This can either be done in iSeries Navigator or using an OS/400 command following these steps:

1. Ensure that clustering is started. On an OS/400 command line, enter the Add Device Domain Entry (**ADDDEVDMNE**) command, and press Enter or F4. You need to enter the following parameters:
   - Cluster (CLUSTER)
   - Device domain (DEVDMN)
   - Node identifier (NODE)
2. In iSeries Navigator, select **Management Central** → **Clusters** → **Cluster_Name** → **Nodes**. Right-click the first cluster node and select Properties. Select the Clustering tab. Specify the device domain and click **OK**. Repeat this for all nodes (using the same device domain name) in the cluster.

Creating a cluster resource group

To create a cluster resource group:

1. Under Management Central, expand **Clusters** → ***Cluster Name*** (ITSOCLU in our example), then right-click **Switchable Hardware** and select **New Group**, as shown in Figure 7-12.

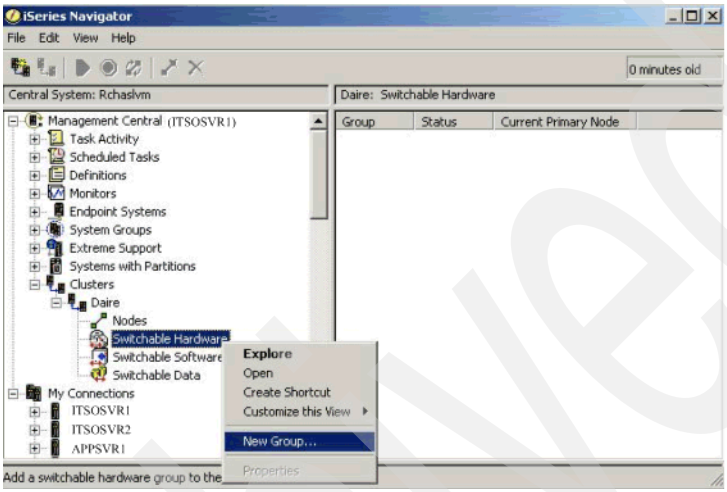


Figure 7-12 Create new cluster resource group

2. Select **ITSOSVR1** as the Primary Node in CRG, and click **Next**.
3. Specify a name for the CRG. We have chosen DEVCRG. Enter the Takeover IP address (192.168.100.150) that you created in “TCP/IP requirements” on page 232 into the appropriate field. Then, click **Next**.
4. You are now asked if you would like to create a new switchable disk pool or if you would like to use an existing one. We are using the Trade IASP that we created in “Creating the IASP” on page 227. Enter the name of the IASP and click **Enter**.
5. Confirm the settings on the summary screen and click **Finish** when you are satisfied.

The cluster resource group that we just configured will manage control of the disk group by ensuring that only one copy is the primary at any given time.
Figure 7-13 on page 238 shows the configuration at this time. Do not start the CRG yet!

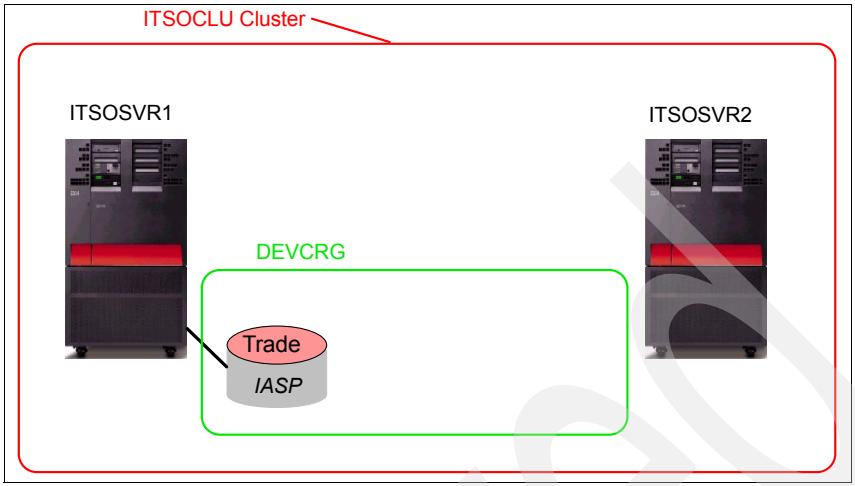


Figure 7-13 Sample configuration after creating the cluster resource group

Configuring the recovery domain and data ports

To configure the recovery domain and data ports, follow these steps:

1. Right-click the **DEVCRG** cluster resource group just created, and select **Properties**.
2. Select the Recovery Domain tab.
3. Specify the primary node and backup node roles.

> **Note:** The Replicate option is used to tell clustering that this node is part of the cluster only for replication purposes. It can never become the primary node. Do not select the Replicate option.

4. Add a site name and data port address for each node by selecting each node and clicking **Edit**. These fields are used for cross-site mirroring, which we will configure more specifically later.
   a. Add a site name for the node. The site contains a subset of recovery domain nodes in the physical location. All nodes at a site have access to the same copy of the auxiliary storage pool. For our example, use SITE1 for the primary node Site name.
   b. Click **Add** to add the Data port IP address. The data port IP address is used to send updates from the source node that owns the production copy of the ASP to the mirror copy on the backup or target node.

      In our case, the IP address that we are using for the primary node is 192.168.100.201. The address used for the backup node is

192.168.100.202. Click **OK** after making the updates for the primary node. Figure 7-14 shows an example.

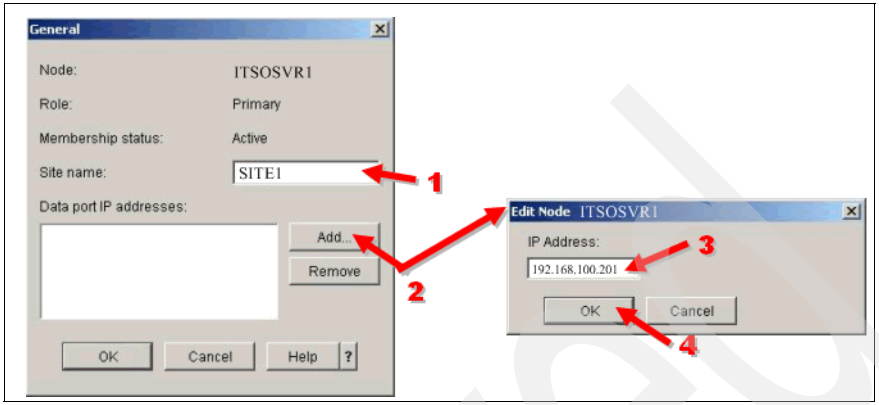


Figure 7-14 Configure the Site name and Data port IP addresses for mirroring

c. Repeat step b on page 238 for each backup node in the domain.

Note: You should define at least two TCP/IP connections between the two sites for the cross-site mirroring replication processing. This provides not only network redundancy but also better performance.

5. Figure 7-15 shows how the DEVCRG properties page looks after making the updates. When you are finished, click **OK**.

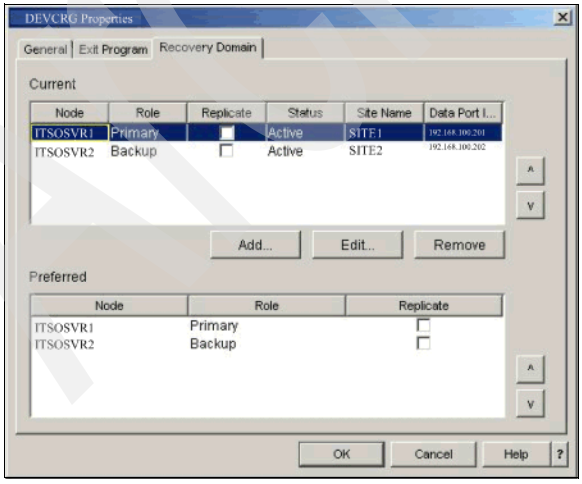


Figure 7-15 Specifying primary and backup nodes in a CRG

Summary

The diagram in Figure 7-16 shows the current configuration after creating the device domain. Note that there is still only one copy of the IASP. The mirror copy is created next when we configure cross-site mirroring. Do not start the CRG at this time!

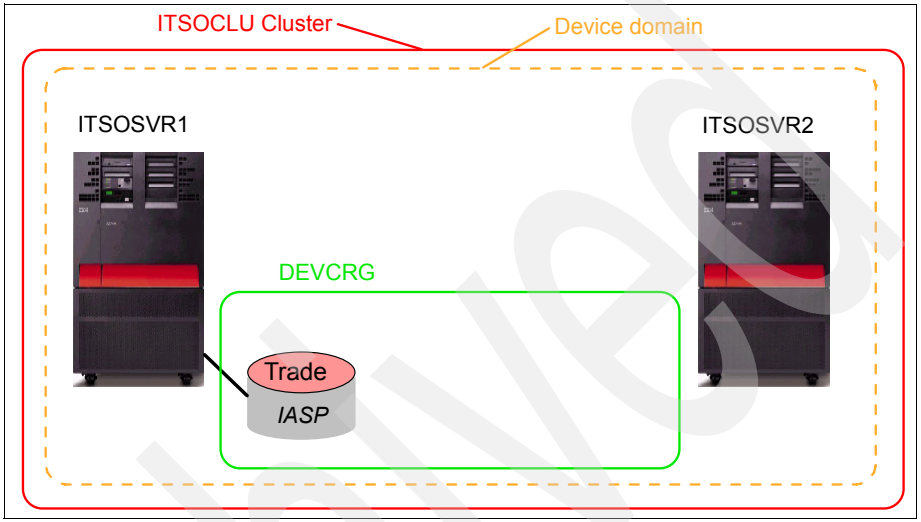


Figure 7-16 Sample configuration after recovery domain creation

7.2.3 Configuring cross-site mirroring

There are several prerequisites for configuring cross-site mirroring. To find the current list of recommended and required fixes, go to:

`http://www-912.ibm.com/s_dir/slkbase.nsf/ibmscdirect/`

Then, click the link for **Geographic Mirroring**.

To configure cross-site mirroring perform the following steps:

1. From the iSeries Navigator, select your primary server. Then, choose **Configuration and Service** → **Hardware** → **Disk Units**.
2. Enter the Service tools user ID and password if you are prompted to do so. Then, click **Disk pools**.
3. Right-click your IASP name, and under the **Geographic Mirroring** menu select **Configure Geographic Mirroring**. See Figure 7-17 on page 241 for details.

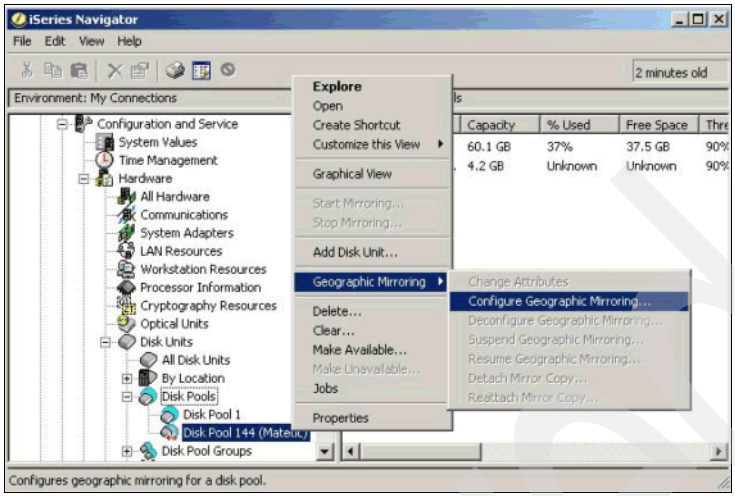


Figure 7-17 Configure cross-site mirroring

4. Click **OK** on the dialog box that confirms that you are starting the configuration of geographic mirroring on the DEVCRG resource group.
5. On the Welcome page confirm that your IASP is in the box and press **Next**.
6. Select your IASP on the next panel, and click **Edit** to change its attributes.
7. Modify the Trade disk pool attributes to the following values:
   a. Update the mode to **Synchronous.**
   b. If you plan to configure local protection check the box to **Protect the data in this disk pool**. This ensures that you have the option to configure local mirroring or parity protection in a few minutes.
   c. Select **OK** to confirm your attribute changes.

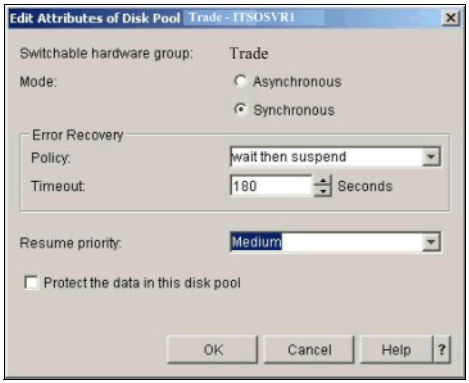


Figure 7-18 Modify attributes for Trade disk pool

8. After returning to the Disk Pools window, click **Next** to continue.
9. Specify the remote node on which you are creating the mirror copy IASP. In our example, we use ITSOSVR2. Click **Next**.
10. It is now time to configure the second IASP by adding unconfigured disk units to the device. Click **Add Disk Units**, select all of the disks to add to the IASP, and click **OK**.
11. Click **Finish** to perform the configuration. This likely takes a few minutes. Also, when the IASP is varied on later, any protection you configured takes effect, which lengthens the time for the vary on.

Starting the CRG

You have now completed the setup for clustering and mirroring and are ready to start the CRG by issuing the `STRCRG ITSOCLU DEVCRG` command or by right-clicking the cluster resource group in iSeries Navigator and selecting **Start**. You should then verify that mirroring started. If it did not start, right-click the disk pool, and under the **Geographic Mirroring** menu, choose **Start Mirroring**.

Summary

Figure 7-19 depicts the completed configuration including the second IASP and cross-site mirroring.

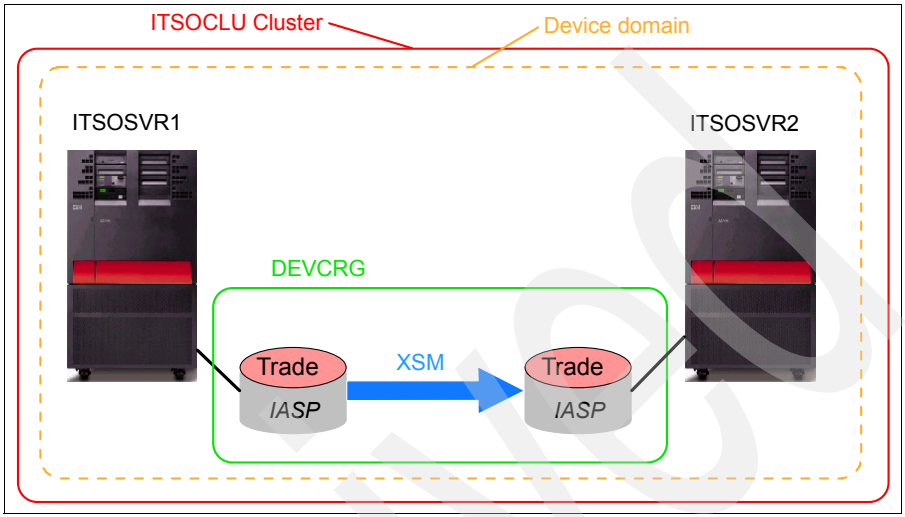


Figure 7-19 Completed clustering and cross-site mirroring configuration

For more detailed information about configuring IASPs, geographic or cross-site mirroring or managing clusters, refer to the iSeries Information Center available at:

`http://publib.boulder.ibm.com/html/as400/infocenter.html`

7.2.4 Restoring the WebSphere application database into the independent ASP

The process for restoring data is not different when using an independent ASP and clustering except that you should take care to specify the destination or *restore to ASP*. All restore commands now have the option of specifying that the data should be restored to a given ASP, which can be referenced by the IASP name or the ASP number. This includes the RSTLIB command to restore full libraries.

Restoring IFS files using the RST command is slightly different in that the destination path you specify should use the UDFS path that is automatically mounted when the IASP is varied on. In our previous example, the IASP was named Trade, so the restore path is /trade/*Subdirectory_Name*.

When configuring mirroring these changes are automatically pushed to the mirror copy as well if it is already started - or it will be copied as part of synchronization if you choose to configure mirroring after restoring all of your data.

7.2.5 Creating a J2C authentication alias

This is an optional step which allows you to store a user ID and an encrypted password as part of the configuration. When this information is needed for configuration objects later, such as in a data source or connection pool, you can choose to use the authentication alias instead of entering the user and password. This has multiple benefits including a single location to make changes if the password were to change and to keep from making this password visible in configuration or properties files.

Follow these steps to configure a J2C authentication data entry in the WebSphere Administrative Console:

1. Select **Security**, then **Global security.** Expand the **JAAS Configuration** section, and select **J2C Authentication data**.
2. Click **New**, and specify the following information to create the authentication data. When completed, the authentication information should be similar to Figure 7-20 on page 245.
   a. Enter an Alias name for the data entry. This should be a descriptive name that you will use in your configurations to identify that this is the authentication data that should be used.
   b. Enter the User ID.
   c. Enter the Password.
   d. Click **OK** when finished.

This authentication entry can now be used wherever you are allowed to select a component-managed authentication alias. We will be using it when creating a data source.

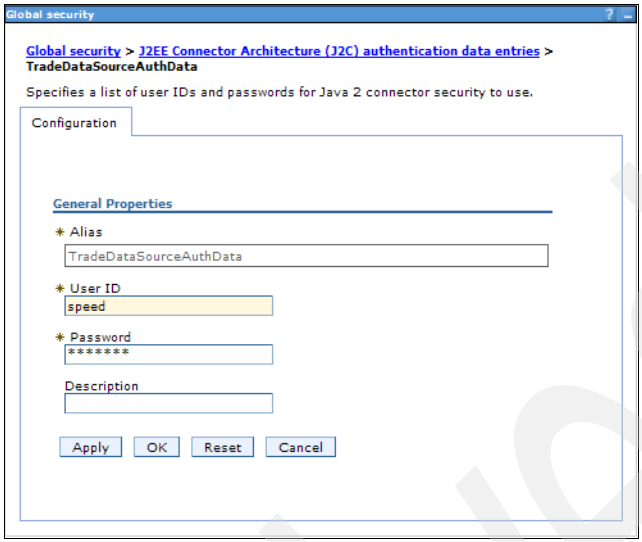


Figure 7-20 Configure a J2C authentication data entry

7.2.6 WebSphere data source configuration

After you have created or restored your database you need to configure a data source in WebSphere so that your Web applications can easily access the database. There are two reasons for the way we configure the data source:

- We are using an independent ASP for our main application data storage.
- We are using clustering so the primary copy of the database could switch to a different system.

Although the database appears local to one WebSphere node, it is a remote database for the rest. Plus, during a failure, the database could switch to a different machine. So, we are going to handle that event in the pool by configuring all connections on all machines as remote database connections, even if the database is on the local system. Then, we configure and direct the connections to the IP takeover address so that when a switch happens, the pool rebuilds the connections to the new database host, which assumes ownership of the IP takeover address.

Creating the JDBC provider

The first step in creating the connection pool is to create a new JDBC™ provider:

1. In the Administrative Console navigation tree, expand **Resources**.
2. Click **JDBC Providers**, and make sure the red arrow is pointing to the scope in which you would like your new provider and data source to be (to switch to

cell level you must remove the data in the node level text box and then click **Apply**).

3. To create a new provider, click **New**.
4. In the General Properties pane there are several fields you need to fill in:
   a. Select the database type of **DB2.**
   b. Select the provider type of **DB2 UDB for iSeries (Toolbox)**.

      The toolbox is the recommended driver if you are connecting to remote databases. Remember that the database can be local for the system currently but after a failover it will be available on a different system as a remote database. Also, because this new data source is published at cell level, each of the other systems views the database as remote. For this reason we are treating all connections such as a remote database. This also allows the pool to automatically reconnect to the database after a failover.

   c. Select the implementation type of **XA data source**. This provides support for 2-Phase Commit if necessary.

   Figure 7-21 shows an example of what you should select for your new provider. Click **Next** when you are finished.

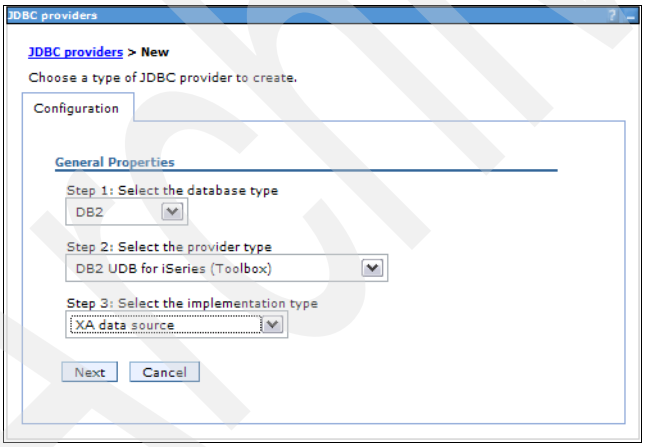


Figure 7-21 Configure a new JDBC provider

5. The next page allows you to change the classpaths and implementation details for the provider. Typically the defaults are fine unless you choose to give it a different name. Click **Apply** to create the provider and to enable the Additional Properties links.

Creating the data source

Next, we create a data source using the provider we just created:

1. The Additional Properties should now be enabled on the right side of page. Click **Data sources**.
2. Click **New** to create the new data source.
3. Optionally change the Name and Description of your data source. In the Component-managed authentication alias field select the authentication alias that you have configured in 7.2.5, “Creating a J2C authentication alias” on page 244.

Figure 7-22 Configure authentication alias for the data source

4. At the bottom of the page, in the Server name field of the DB2 UDB for iSeries (Toolbox) data source properties frame, enter the takeover IP address that you assigned to the cluster resource group. This makes sure the database can be accessed regardless of whichever system is the current primary for the database. In our example, the takeover IP address is 192.168.100.150.

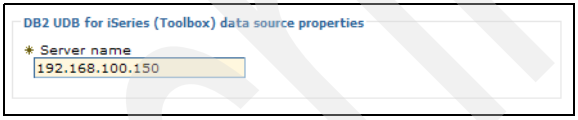


Figure 7-23 Specifying the IP takeover address for the data source

5. Click **Apply**. This enables the links under Additional Properties.
6. Select **Connection pool properties** from the Additional Properties. Because we are using an independent ASP we want our pool to rebuild all of the connections in the event of a failover to the backup system. Optionally, you can do this by changing the Purge policy field to **Entire pool**. Then click **OK.**
7. There are also several custom properties that you must configure (and some optional ones) for use with a failover database or IASP environment. From your data source page, click **Custom properties** and change or add each of the following properties as needed:
   - **databaseName**: The name of the database that you are connecting to that lives in the IASP.
   - **libraries**: This field allows you to specify the name of the library the data for your application is in.

- **keepAlive**: Set this field to **true** so that TCP probes will be sent periodically to test the liveness of the other end of the connection. This is critical in order for connections to get closed in the event of a back-end database failure so that WebSphere threads are available to establish new connections when the IP takeover address becomes active at the end of database server failover processing.
- **receiveBufferSize**: Set the Value to 1000000.
- **sendBufferSize**: Set the Value to 1000000.

Note: You can find detailed descriptions of these custom properties and the steps for configuring a data source for use in a WebSphere on the iSeries environment in the paper *iSeries and High Availability An e-Business Perspective*, which is available at:

`http://www.ibm.com/servers/eserver/iseries/software/websphere/wsappserver/product/iSeriesAndHa.pdf`

8. **Save** the new settings.

7.2.7 Messaging engine datastore

The default messaging provider support requires a separate datastore for every messaging engine (ME). Each ME stores its configuration properties in its assigned datastore. This datastore is the equivalent of a collection in DB2 or a library on iSeries systems and must be accessible from every server that could potentially run the messaging engine.

Because we are providing failover capability we should to configure WebSphere to keep this datastore also in the independent ASP for high availability and failover capability. To do this, you first need to create the collection on the independent ASP and then configure WebSphere to use it.

Creating the ME datastore in the IASP

Follow these steps to create the datastore in the IASP:

1. First you must obtain the name of the collection you need to create. You can find this information in the Administrative Console by clicking Service integration → **Buses** → ***ClusterName*** → **Messaging engines** → ***ClusterNodeName*** → **Data store**.

   Write down the Schema names, for example IBMME2, as this is the name of the collection you need to create later. See Figure 7-24 on page 249.

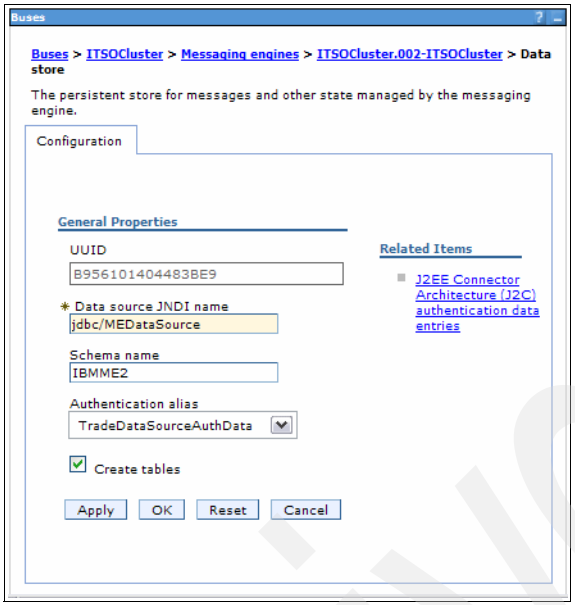


Figure 7-24 Messaging engine data store properties

2. You need to create the collection using an SQL command. Simply creating the library on the system would not provide you with all of the files that are created when you make a new collection.

   Enter STRSQL on an OS/400 command line.

3. You are, by default, connected to the system database. To connect to the IASP enter the following:

   ```
   connect to iasp_Name using userid
   ```

   For example, our command is **`connect to trade using qsecofr`**.

   Complete and submit the password when prompted.

4. Create the collections using the following:

   ```
   create collection Schema_Name
   ```

   In this command, *Schema_Name* is the name of the ME datastore that you noted in step 1 on page 248, for example, IBMME2.

5. Exit the SQL command entry.

6. Verify that the data source used for the ME datastore points to the correct location in the IASP:
   - In the Administrative Console navigation tree, click **Resources → JDBC Providers → DB2 JDBC for iSeries (Toolbox) → Data sources → MEDataSource**.
   - Verify or enter the takeover IP address that you assigned to the cluster resource group into the DB2 UDB for iSeries (Toolbox) data source properties frame. Compare to “Creating the data source” on page 247.
   - Save if you made any changes.

The messaging engine now uses the database in the independent ASP for saving its data. This data then continues to be available on the backup machine in the event of a failover or system outage.

7.2.8 Configuring iSeries TCP/IP settings

You should also configure some TCP/IP settings for the host, using the **CHGTCPA** command, to achieve a timely failover for your database and connections. Specifically, you should verify the following attributes, as shown in Example 7-1 on page 251:

- TCP keep alive: 2 minutes
- TCP R1 retransmission count: 3
- TCP R2 retransmission count: 9
- ARP cache timeout: 2 minutes

Although you might need to make some additional adjustments for each specific configuration, these settings are a good starting point for optimizing the timeouts to handle a database failover.

Example 7-1 iSeries TCP/IP configuration values

```
Change TCP/IP Attributes (CHGTCPA)
Type choices, press Enter.
TCP keep alive . . . . . .   2            1-40320, *SAME, *DFT
TCP urgent pointer . . . . . .   *BSD          *SAME, *BSD, *RFC
TCP receive buffer size  . . . .   8192         512-8388608, *SAME, *DFT
TCP send buffer size . . . . . .   8192         512-8388608, *SAME, *DFT
TCP R1 retransmission count  . .   3            1-15, *SAME, *DFT
TCP R2 retransmission count  . .   9            2-16, *SAME, *DFT
TCP minimum retransmit time  . .   250          100-1000, *SAME, *DFT
TCP closed timewait timeout  . .   120          0-14400, *SAME, *DFT
TCP close connection message . .   *THRESHOLD   *SAME, *THRESHOLD, *ALL...
UDP checksum . . . . . .   *YES          *SAME, *YES, *NO
Path MTU discovery:
   Enablement . . . . . .   *YES          *SAME, *DFT, *NO, *YES
   Interval . . . . . .   10           5-40320, *ONCE
 IP datagram forwarding . . . . .   *YES          *SAME, *YES, *NO
 IP source routing  . . . . . .   *YES          *SAME, *YES, *NO
 IP reassembly time-out . . . . .   10            5-120, *SAME, *DFT
                                                                More...
 F3=Exit   F4=Prompt   F5=Refresh   F10=Additional parameters   F12=Cancel
 F13=How to use this display       F24=More keys
```

7.3 Transaction Manager configuration

Many tips for configuring the Transaction Manager for high availability are found in 6.7, “Transaction Manager high availability” on page 201. However, there are a few differences to this in iSeries environments. Several of the tips in this section count on having a shared file system for sharing information and the iSeries integrated file system is not yet compatible with NFS V4.

One solution is to use the QNTC file system for sharing files in the iSeries environment. This means that we support an i5/OS™ WebSphere cluster as well as a heterogeneous WebSphere cluster containing both Windows and i5/OS cluster members when Transaction Manager logs are part of the configuration.

This section explains the high level steps for configuring high availability for the transaction logs using QNTC. For more details on configuring the transaction logs see 6.7, “Transaction Manager high availability” on page 201.

Before we configure the file system, there are a couple of important points to consider in order to ensure back-end failures are handled properly. First, we recommend that you use the same IASP to host both the transaction logs and the associated application databases. This ensures that both are made available at the same time during a failover event in order to avoid rollback processing. Journaling of access paths is also critical so that the application databases are truly available for reading and writing immediately after the IP takeover address becomes active. Following are the steps to configure the QNTC file system for the transaction logs.

Creating QNTC links

First, you need to create the QNTC links and directories on the IASP:

1. Enter **WRKLNK** on an OS/400 command line. Then change your current directory to QNTC using the **cd qntc** command.
2. Because we are creating the log files on the IASP, which could switch to a different system, we need to create a link to the IP takeover address which points to the current primary. The command is:

   ```
   MKDIR Takeover_IP_Address
   ```

   In our case the command is:

   ```
   MKDIR 192.168.100.150
   ```

   Important: You must create this directory on each of the cluster member systems.

   Also, the directories that are added by the **mkdir** command only remain visible until the next IPL of the iSeries servers that host the WebSphere cluster members. Therefore, you need to reissue the commands listed in steps 1 and 2 after an IPL of those systems to ensure WebSphere access to the transaction logs that are hosted on a remote iSeries server. As an alternative, we recommend that a small script be written to issue these commands automatically after each IPL.

   Another important tip is with regard to the remote file server systems. Change the time-out period property for iSeries NetServer™ on the two i5/OS cluster node systems to *not* allow idle connections to be terminated by NetServer. To change this setting:

   1. In iSeries Navigator, open the iSeries NetServer Properties panel.
   2. Go to the Advanced tab.
   3. Click **Next start** and deselect **Allow inactive sessions to time-out**.
   4. Click **OK**.
   5. Restart iSeries NetServer to pick up the change.

6. Change into the new directory and you should now see the IFS of the current primary system.
7. Note that we are not yet pointing to the IASP. Change the directory to the IASP using

   ```
   cd 'root/iasp_Name'
   ```

   In our case the IASP is named trade so the command is:

   ```
   cd 'root/trade'
   ```

8. Create a new directory called logs that will hold the transaction logs of all WebSphere cluster members:

   ```
   MKDIR logs
   ```

9. Change into the logs directory (cd logs). Then, enter the **WRKLNK** command again to see the contents of the current directory.
10. We now need to create directories for each of the cluster members. This will be the new location for the transaction logs. By naming them the same as the member name, the Transaction Manager is able to ensure that the information can be recovered and used by another member if one fails.

    For each cluster member type `mkdir Member_Name` and press Enter.

11. When all directories for the cluster members are created, you need to copy the content of the existing transaction log directories into the new directories on the IASP.

Configuring transaction log location in WebSphere

The next task is to enable WebSphere for transaction log recovery and to configure the newly created directories on the IASP as the transaction log directories. Follow these steps:

1. Go to the WebSphere Administrative Console and stop the cluster by selecting **Servers** → **Clusters**. Select your cluster and click **Stop**.
2. High availability for persistent services is a cluster setting that needs to be activated. From the clusters page click your cluster name to display the details of your cluster. Select **Enable high availability for persistent services** on the Configuration tab, as shown in Figure 7-25 on page 254. Click **OK**.

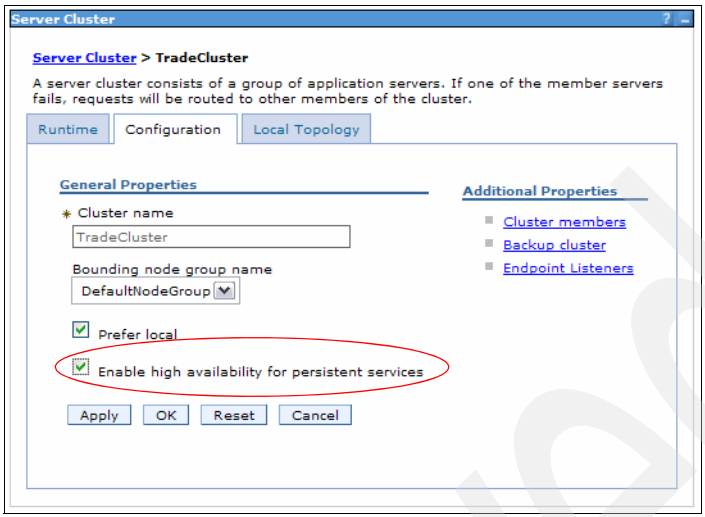


Figure 7-25 Enable high availability for persistent services

3. Change the transaction log directories of all cluster members from within WebSphere to use the new directories on the IASP.

   Click **Servers** → **Application servers** → ***AppServer_Name*** → **Container Services** → **Transaction Service**.

   Enter the new Transaction log directory location as shown in Figure 7-26 on page 255. Our sample path is /qntc/192.168.100.150/root/trade/logs/member4.

   This needs to be repeated for each cluster member.

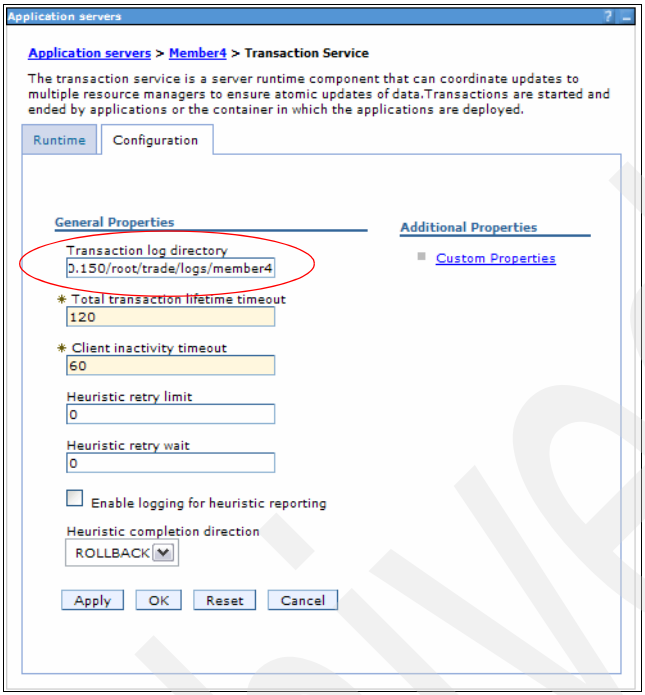


Figure 7-26 Configuration of transaction log location

4. Save and synchronize the configuration.

Authorization

Next, you need to set the proper authorities. The QEJBSVR user profile needs to have authority to view the logs on the remote system. By default, QEJBSVR does not have a password. However, to avoid conflicts, we need to specify the same password for QEJBSVR on each of the WebSphere cluster nodes. Follow these steps:

1. To set the password, invoke the **`WRKUSRPRF QEJBSVR`** command.
2. Enter Option 2 (= change), and then set the User password to a common value on each system.
3. You should also validate that the profile has authority to each of the directories that contain the transaction logs.
4. Repeat steps 1 to 3 on each node in the cluster.

Important: If your IASP is not directly connected to the systems that have WebSphere installed and are running the WebSphere cluster, then you need to create the QEJBSVR user profile with the same password on the database systems. This is because this user profile comes with WebSphere and, thus, is not likely to exist on the database or iSeries cluster systems. Figure 7-27 shows a scenario that illustrates such an environment.

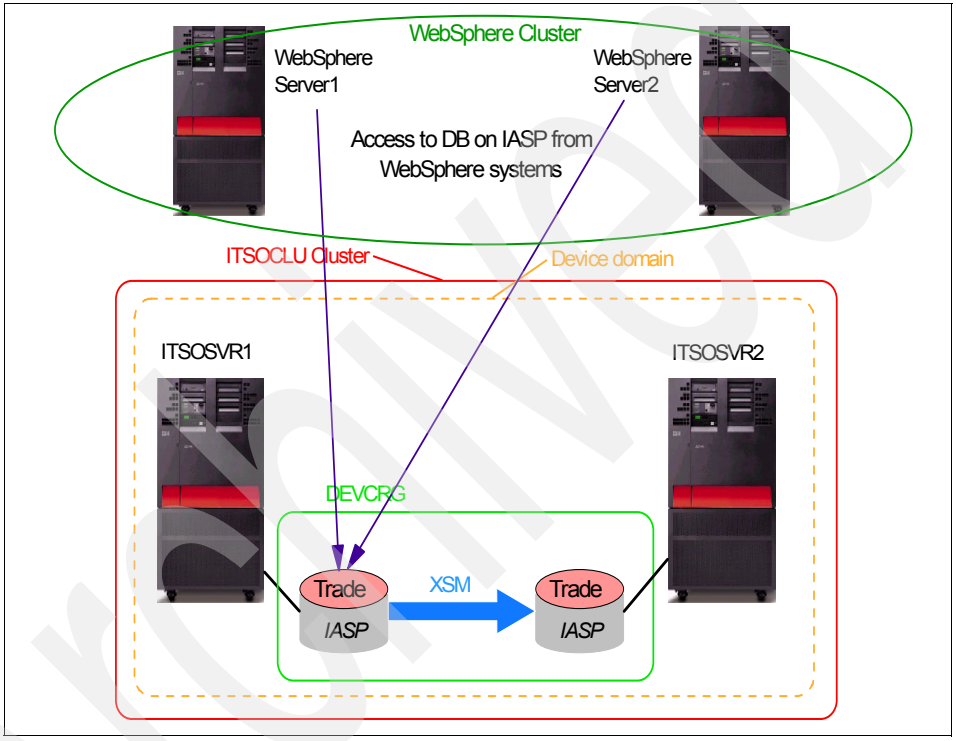


Figure 7-27 WebSphere and database/IASP on different iSeries systems

Starting the WebSphere cluster

Restart the previously stopped WebSphere cluster in **Servers** → **Clusters**. Transaction log recovery is now enabled and transactions are logged in the directories on the IASP.

7.4 Reference material

The following resources are available:

- The white paper (*iSeries and High Availability: An e-Business Perspective*) explains HA options on the iSeries for the various components of an e-business environment, such as firewalls, the Load Balancer and HTTP server, back-end data, and so on. You can find the paper at:

  `http://www.ibm.com/servers/eserver/iseries/software/websphere/wsappserver/product/iSeriesAndHa.pdf`

- *Clustering and IASPs for Higher Availability on the IBM @server iSeries Server*, SG24-5194

8

WebSphere HA on z/OS

This chapter describes WebSphere high availability on zSeries. Its focus is on z/OS systems.

WebSphere Application Server for z/OS takes advantage of some of the high availability features of z/OS available in a Parallel Sysplex® environment. This chapter describes how some of these high availability functions are achieved on z/OS systems. The topics discussed are:

- zSeries Parallel Sysplex
- WebSphere V6.0.1 for z/OS topology overview
- z/OS workload management and WebSphere workload management
- Workload distribution to different systems in a Parallel Sysplex
- Failover options for WebSphere Application Server Version 6 on z/OS
- Transaction failover
- HTTP session and stateful session bean failover
- Messaging engine high availability
- DB2 data sharing
- WebSphere MQ high availability
- A high availability configuration

This chapter does not go into great detail about how to set up WebSphere HA on z/OS. For more information about the different scenarios and set up, see *WebSphere for z/OS V6 High Availability*, SG24-6850.

8.1 zSeries Parallel Sysplex

The zSeries Parallel Sysplex Clustering Technology offers the highest standard for multi-system availability. See Figure 8-1. This section introduces some of the terminology that we use in this chapter.

- LPAR

  LPAR stands for logical partition. A logical partition is a division of zSeries computing resources, including processors, memory and storage, and so forth into multiple sets of resources, so that each set of resources can be operated independently with its own operating systems instances and application. From an application point of view, you can think of a z/OS LPAR as a single system image as with any other operating system. In this chapter, the terms *LPAR* and *system* are used interchangeably.

- Parallel Sysplex

  A sysplex is a group of z/OS systems that communicate and cooperate with one another using specialized hardware and software. They are connected and synchronized through a Sysplex Timer® and enterprise systems connection channels. A Parallel Sysplex is a sysplex that uses one or more coupling facilities. See Figure 8-1.

- Coupling Facility

  A Coupling Facility is a special LPAR accessible by every system in the Parallel Sysplex. It provides data sharing, high-speed caching, list processing, and lock processing for any application on the sysplex.

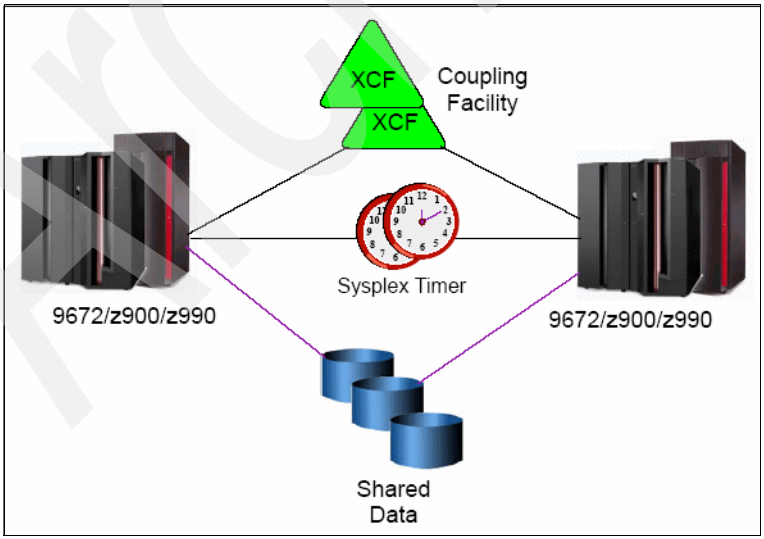


Figure 8-1 Parallel Sysplex architecture elements

8.2 WebSphere V6.0.1 for z/OS topology overview

This section introduces the available WebSphere Application Server for z/OS versions and their implementation on z/OS.

8.2.1 Base application server on z/OS

A base WebSphere Application Server on a non-z/OS platform is a process with a JVM running inside that provides a set of services. These services include an embedded HTTP server, a Web container, an EJB container, a messaging engine, a Web Services engine, and depending on the version of the product, eventually one or more other services.

On z/OS, a base application server has a number of address spaces or processes in UNIX terms. These address spaces are shown in Figure 8-2 and explained in the sections that follow.

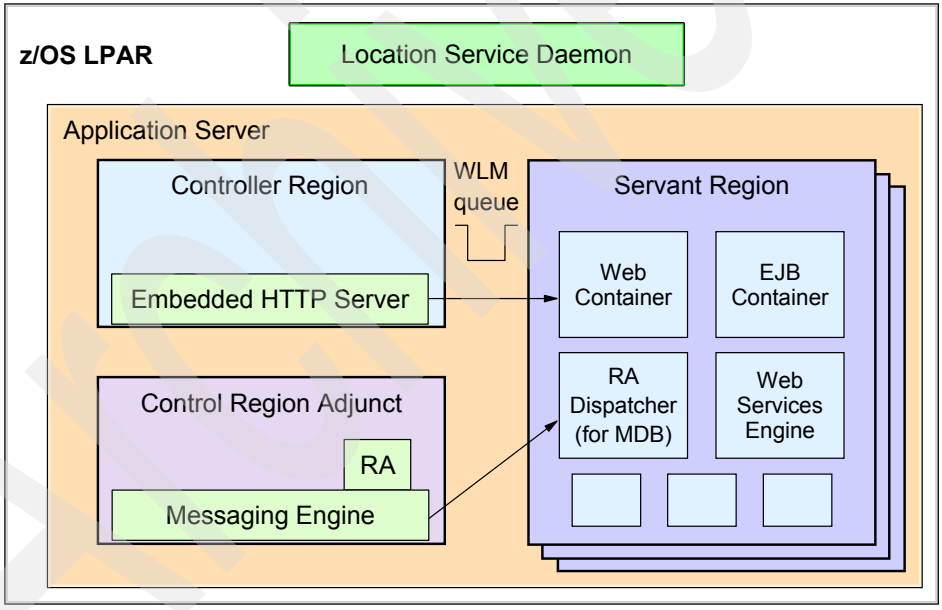


Figure 8-2 Base application server on z/OS

Controller region

One address space is the *controller region* (CR) or *controller*, which manages tasks and handles communications. The embedded HTTP server is also hosted in the CR. The CR is the endpoint of an application server. There is one controller region address space per application server.

Servant region

The second type of address space is the *servant region* (SR) or *servant*. There is a JVM running in each servant region, and this is where the Web container, EJB container, Web Services engine, and so on are running. The servant region has no TCP/IP endpoint. The servant region is where the application code is being processed.

An application server can have one or more servant regions. z/OS workload management (WLM) starts additional servant regions as needed when workload increases. For more information about WLM, see 8.3, "z/OS workload management and WebSphere workload management" on page 265.

Control region adjunct

The third type of address space is new in WebSphere Application Server V6. It is called *control region adjunct* or *CRA*. The CRA is used to host the messaging engine. It acts as a messaging communication endpoint, comparing to the controller region as the HTTP communication endpoint.

Unlike the controller region which is always there, the CRA is only present when the application server is a member of the bus and has a messaging engine created.

Location Service Daemon

In addition to these address spaces, there is an address space called *Location Service Daemon* that runs in each WebSphere LPAR. The Location Service Daemon provides the CORBA location service in support of RMI/IIOP, which is the protocol for remote EJB invocation. Multiple application servers in the same LPAR share the same Location Service Daemon.

8.2.2 Network Deployment on a z/OS LPAR

A Network Deployment cell environment on a single LPAR is shown in Figure 8-3.

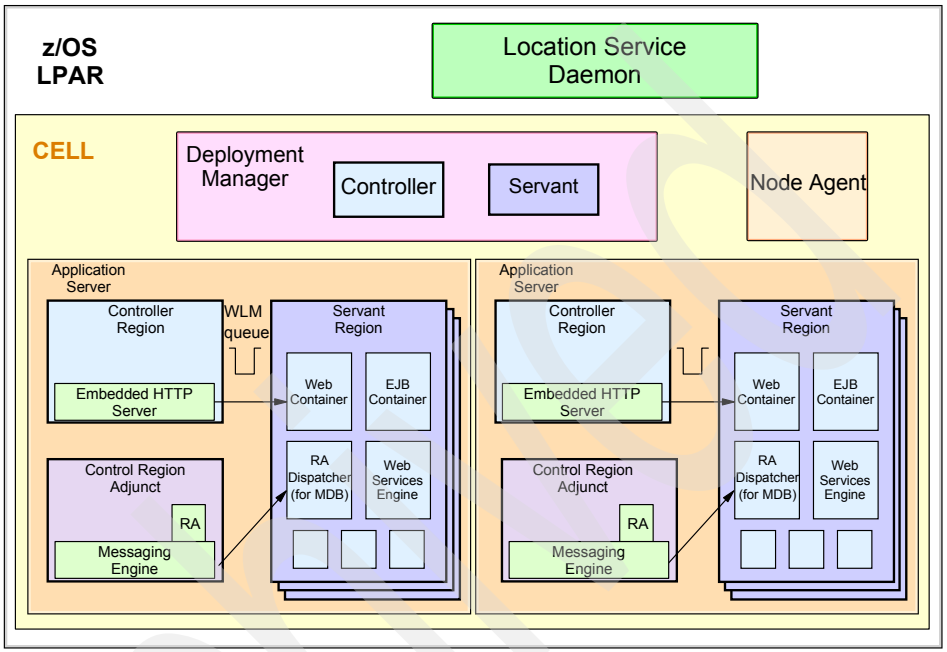


Figure 8-3 Network Deployment on a z/OS LPAR

Deployment Manager

The Deployment Manager also has a pair of address spaces: a controller and a servant. The Administrative Console is a J2EE application that runs in the Deployment Manager servant region. There is only one servant region for the Deployment Manager because some administrative tasks require serialization that can only run in a single JVM.

Node Agent

The Node Agent runs in its own address space.

Location Service Deamon

There is one Location Service Daemon for each cell in each system.

8.2.3 Network Deployment in a Parallel Sysplex environment

Various WebSphere configurations are possible in a Parallel Sysplex environment, as shown in Figure 8-4.

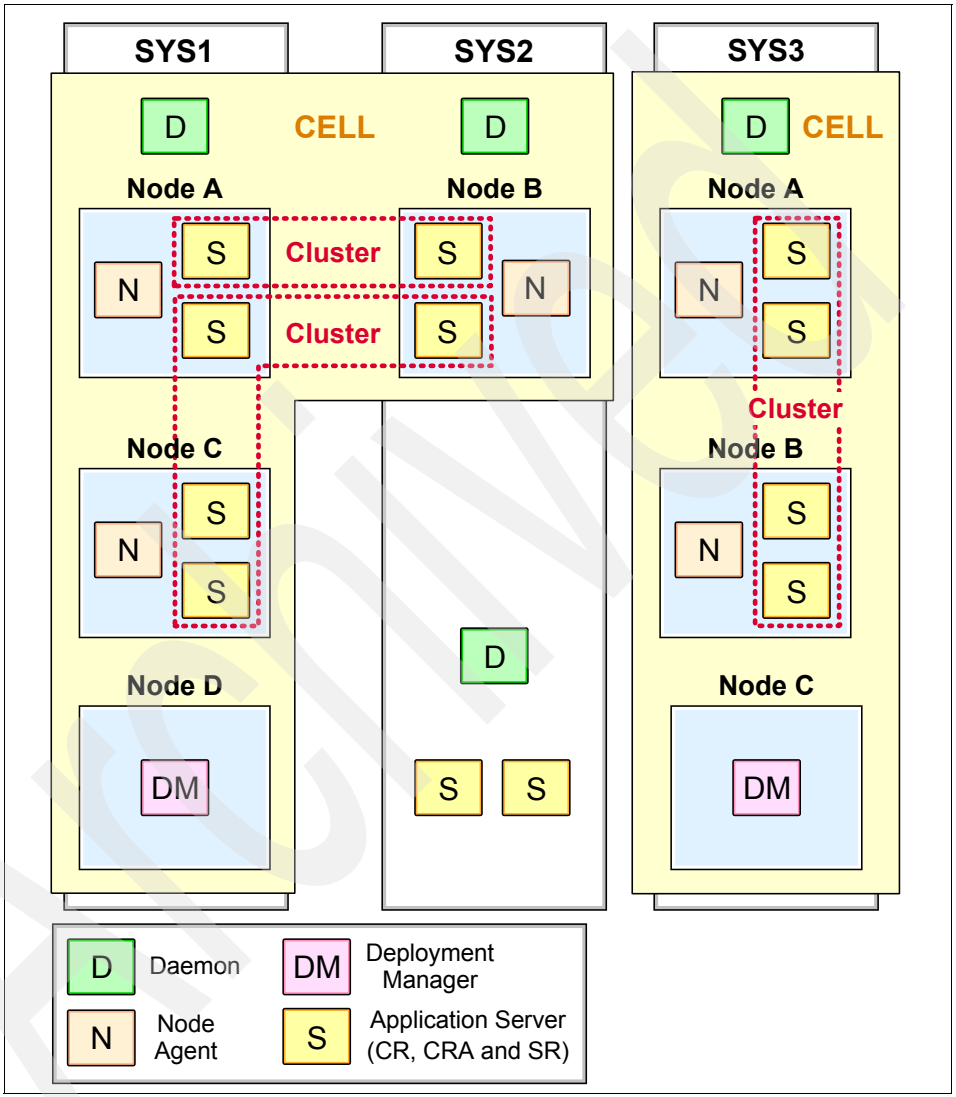


Figure 8-4 WebSphere for z/OS in a sysplex

A WebSphere cell can span across multiple LPARs with one Deployment Manager that resides on one of the LPARs. Thus, a cluster in a cell can include

application servers from multiple LPARs within the same cell. Multiple cells can also coexist in the same LPAR.

8.2.4 Mixed platform cells

A new concept called *node group* is introduced in WebSphere Application Server V6. A node group is a collection of managed nodes. A node group defines the boundary for cluster formation. Nodes organized into a node group should be enough alike in terms of installed software, available resources, and configuration to enable servers on those nodes to host the same applications as part of a server cluster.

With node groups it is possible to mix z/OS WebSphere nodes with distributed WebSphere nodes in the same cell. A distributed node and a z/OS node cannot belong to the same node group; a z/OS node belongs to a Sysplex Node Group, which is unique for the z/OS environment. Because a node group defines the cluster boundary and a distributed and z/OS node cannot be combined in the same cluster in WebSphere Version 6, a WebSphere HA configuration does not include both a distributed and z/OS system.

For more information about node groups see *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

8.3 z/OS workload management and WebSphere workload management

WebSphere Application Server for z/OS takes advantage of several z/OS functions to assure availability. This section explains the z/OS workload management (WLM) concept and how it manages WebSphere.

Workload management for z/OS is different than the workload management component that is included in WebSphere Application Server Version 6 for all platforms. WLM on z/OS provides sysplex-wide workload management capabilities based on installation specified performance goals and the business importance of the workloads. Workload management can dynamically start and stop server address spaces to process work from application environments.

WLM manages the response time and throughput of WebSphere transactions according to their assigned service class, the associated performance objectives, and the availability of system resources.

When starting a WebSphere application server on z/OS, a controller region is first started by MVS™ as a started task, then WLM starts one or more servant

regions. The minimum (at least one) and maximum numbers of servant regions that stay up is configured through the Server Instance settings of the application server. See Figure 8-5.

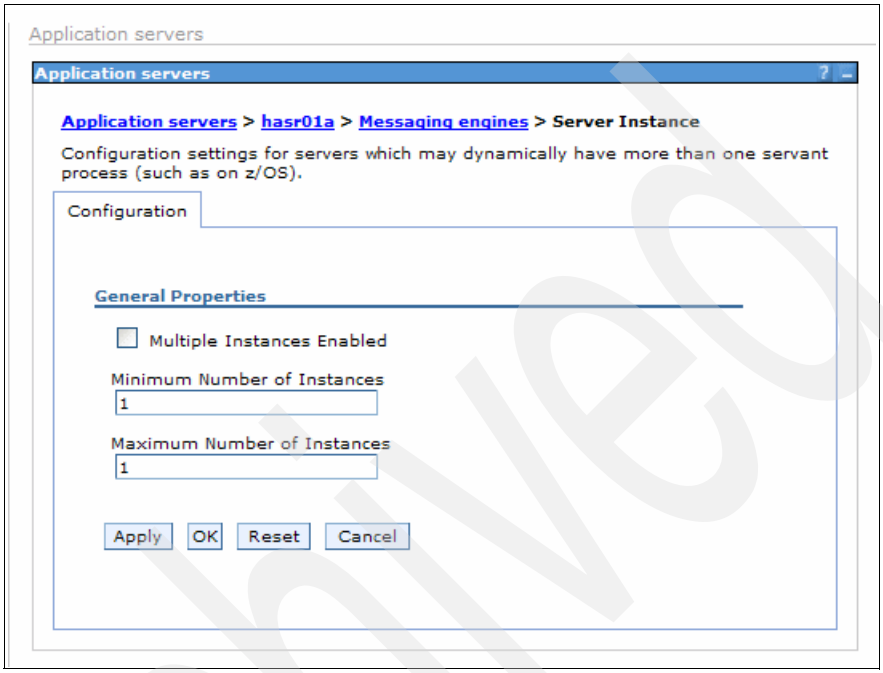


Figure 8-5 Configure minimum and maximum servant regions

When a work request arrives at the controller region, it is put on an in-memory WLM queue. Depending on the priority of the work, WLM dispatches work to worker threads in the servants. As workload increases, WLM might start more servants to meet the WLM performance goal setting for the application server, up to the maximum servant number specified.

WLM supports servant region session affinity for HTTP requests. For applications with no HTTP sessions or short HTTP session requests, you can use the default workload distribution strategy, which picks a hot servant to handle the new request for better performance. A hot servant has just had a recent request dispatched to it and has threads available as well as resources in memory.

For applications with many HTTP sessions and long HTTP sessions, WLM features a function called *even distribution* of HTTP requests. This function is available in z/OS 1.4 and later. Even distribution of HTTP requests supports distribution of incoming HTTP requests without servant affinity in a round-robin manner across the servants. When a new HTTP request without affinity arrives

on a work queue, WLM dispatches the request to a servant with available worker threads and the smallest number of affinities.

From an availability point of view, WLM ensures that the configured minimum number of servants are available to handle requests. If for some unexpected reason one servant goes down, WLM starts another one under the same controller. The application server stays up. This is different than WebSphere Application Server on other platforms. When the JVM of an application server on non-zOS platform goes away, the application server goes away and needs to be restarted (automatically or manually, depending on its configuration).

WebSphere Application Server for z/OS supports the use of HTTP session objects in memory for application servers with multiple servants. WLM dispatches work to servants with session affinity.

In a Parallel Sysplex environment, WLM provides system resource information to subsystems such as DB2 and WebSphere so they can properly distribute work. With WebSphere Application Server for z/OS, WLM helps the Sysplex Distributor (see 8.4, “Distributing HTTP and IIOP requests to different systems within a Parallel Sysplex” on page 268) to balance and route requests to different LPARs for IIOP requests and stateless HTTP request.

Again, WLM on z/OS is different and complementary to the workload management functions included in WebSphere Application Server for other platforms. Together they help to achieve high availability of WebSphere Application Server on z/OS.

Table 8-1 lists the high level differences between WLM on z/OS and WebSphere WLM.

Table 8-1 High level comparison of WLM on z/OS and WebSphere WLM

	Workload management for z/OS	**WebSphere Application Server V6 workload management**
Product	Part of z/OS	Part of WebSphere Application Server version 6
Management mechanism	System resource management, in-memory work requests routing	Application level distribution and workload balancing
Workload distribution	Distributes requests to different servants within an application server	Distributes requests to application servers within a cluster
Management level	Manages within each LPAR (system) Also provides capacity information of each system to Sysplex Distributor for workload routing among multiple systems within a sysplex	Manages application servers within a cluster across multiple systems
WebSphere configuration requirement	Any base server or application server within a cluster	Cluster

In summary, WLM on z/OS allows you to classify and prioritize work so that performance goals are respected and so that resources are given to the proper workload.

8.4 Distributing HTTP and IIOP requests to different systems within a Parallel Sysplex

Distributing TCP/IP workload among systems within a Parallel Sysplex can be achieved through the Sysplex Distributor, which is built on Dynamic virtual IP addresses (DVIPA). Dynamic virtual IP addressing (DVIPA) is a z/OS Communication Server function that is used in high availability configuration in sysplex.

The Virtual IP Address, or VIPA, provides an IP address that is owned by a TCP/IP stack but that is not associated with any particular physical adapter. Because the VIPA is associated with a virtual device, it is always available as

long as its owning TCP/IP stack is functional. For systems such as zSeries with multiple network adapters, VIPA provides failure independence from any particular adapter or link, as long as at least one is available and connected. VIPA becomes unavailable when its owning TCP/IP fails.

DVIPA improves the VIPA function by allowing a TCP/IP stack to be moved from one system to another in the sysplex. DVIPA takeover is possible when a DVIPA is configured as active on one stack and as backup on another stack within the sysplex. When the stack on which the DVIPA is active terminates, then the backup stack will automatically activate the DVIPA and notify the routing daemon. For DVIPA takeover to be useful, the applications that service the DVIPA addresses must be available on the backup stacks.

DVIPA can be used by the Deployment Manager and by application servers.

8.4.1 Sysplex Distributor

The Sysplex Distributor is a z/OS Communication Server function that provides a single visible IP address to the network for the entire Parallel Sysplex and balances TCP/IP workload within the sysplex.

The Sysplex Distributor consists of a primary distributor stack denoted by a primary DVIPA (possibly a backup stack VIPA) and a set of target stacks. An inbound packet destined for that DVIPA flows through the primary distributor stack which then forwards the packet over an internal link to the selected target stack. Only inbound requests go through the distributor stack. The distributor consults other system information, including capacity information provided by WLM, to make the routing decision. Figure 8-6 on page 270 shows Sysplex Distributor in a Parallel Sysplex environment.

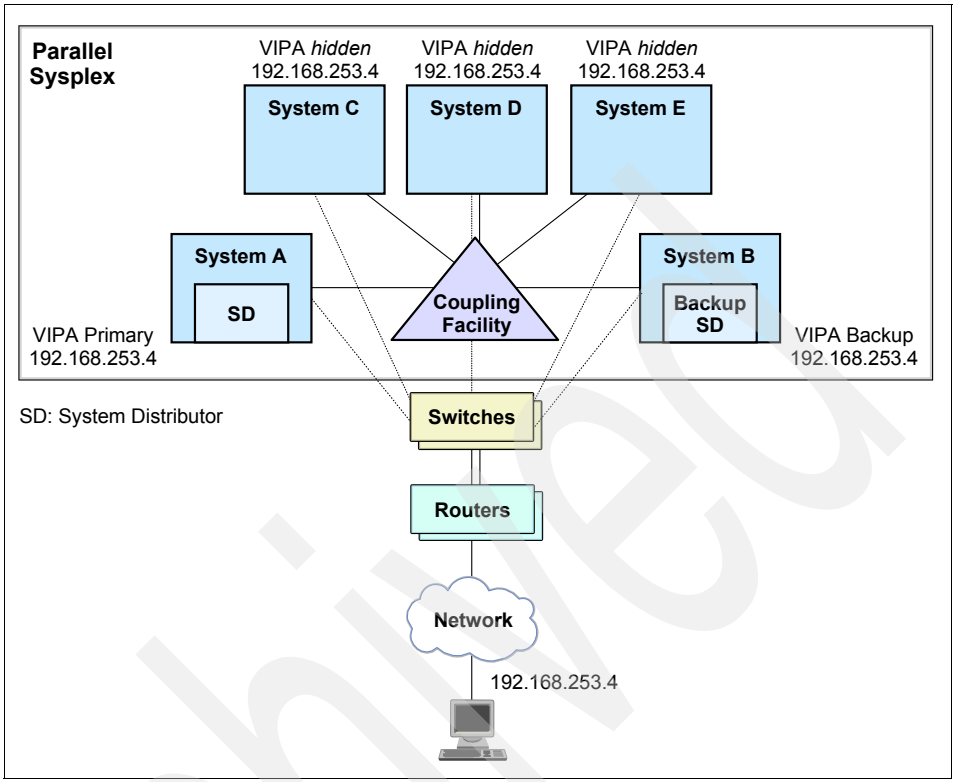


Figure 8-6 Sysplex Distributor

Note: The IBM-recommended implementation is that if you are running in a sysplex, set up your TCP/IP network with Sysplex Distributor to increase availability and aid in workload balancing.

Sysplex Distributor routes and balances TCP/IP requests. It is not capable of making content-based decision. It acts as a secondary workload balancer in a WebSphere environment, located behind the HTTP plug-in of the HTTP server on z/OS or on other platforms. It balances IIOP requests and sessionless HTTP requests. In a Web application serving environment, Sysplex Distributor is complementary to other balancing techniques such as CISCO and WebSphere Application Server Edge Components.

8.5 Failover options for WebSphere Application Server V6 on z/OS

Before V6, WebSphere Application Server for z/OS used the z/OS Automatic Restart Manager (ARM) and Peer Restart and Recovery (PRR) for failover. This option still exists in V6, in addition to the new WebSphere HAManager peer-recovery option.

8.5.1 ARM and PRR

The z/OS ARM enables fast recovery of the subsystems that might hold critical resources at the time of failure. It restarts a started task and job automatically after it detects their failure.

WebSphere Application Server for z/OS uses ARM to recover application servers. Each WebSphere Application Server for z/OS controller registers with an ARM group. If the controller terminates abnormally, ARM tries to restart the failing address space and dependent address spaces in appropriate order on the same system. This recovery mechanism applies to the application servers, Deployment Manager and Node Agents.

ARM can also use another function called Peer Restart and Recovery (PRR) for WebSphere Application Server V6 on z/OS to restart a related server on an alternate system in the same cell.

PRR restarts the controller on another system and goes through the transaction restart and recovery process so that we can assign outcomes to transactions that were in progress at the time of failure. During this transaction restart and recovery process, data might be temporarily inaccessible until the recovery process is complete. The restart and recovery process does not result in lost data. PRR does not require a cluster configuration but it does require a node to be designated as a Recovery Node, and that the Location Service Daemon and Node Agent are both running on that system.

8.5.2 High Availability manager (HAManager)

Failover with ARM and PRR is a cold failover. With the introduction of the WebSphere HAManager in WebSphere Application Server V6, you have the option of hot standby and peer recovery. It does not restart a server. Instead, the transaction on the failing server is recovered by a different application server in the same cluster. For more information about HAManager and peer recovery, see Chapter 6, “WebSphere HAManager” on page 175.

Only the Type 4 JDBC driver is supported by the HAManager. HAManager is required for Web Services, the default messaging provider, JCA 1.5 connectors and so on. It is recommended to gradually migrate to the HAManager approach because the PRR function will not be supported in future releases of WebSphere Application Server on z/OS.

Important: You can use either HAManager *or* PRR for recovery in WebSphere V6. However, do not use both, because unpredictable results can occur.

8.6 Transaction logging and recovery

WebSphere Application Server V6 on z/OS supports two types of transaction recovery: Resource Recovery Services (RRS), part of z/OS, and XA transactions.

8.6.1 A word on 2-Phase Commit (2PC)

Applications that need to coordinate updates with more than one resource manager can use the two-phase commit process to do so. For each unit of recovery (UR) there is one sync point manager and a number of resource managers. The sync point manager determines the outcome, either commit or backout for each UR.

If the resource manager cannot commit the changes, it tells RRS to back out the changes. If all the resource managers agree to commit, RRS stores the decision in an RRS log, which means the decision is hardened, and phase 2 begins.

When the commit decision is agreed to, the application changes are considered to be committed. If the application, any of the involved resource managers, RRS, or the system fails after the decision is hardened, the application changes are made during restart. If the decision is to back out the changes, RRS generally does not harden the decision, and phase 2 begins as soon as the decision is made.

8.6.2 RRS

Resource Recovery Services (RRS) is a system-wide (read LPAR-wide) resource recovery subsystem included in z/OS. It provides two-phase commit support across participating resource managers. RRS-compliant resource managers include WebSphere Application Server, DB2, CICS®, IMS™, and WebSphere MQ.

RRS must be up and running before the WebSphere Application Server for z/OS servers are started. RRS plays a key role in Peer Restart and Recovery (PRR).

WebSphere Application Server for z/OS supports resource adapters (RAs) that use Resource Recovery Services (RRS) to enable global transaction processing, in addition to XA and non-XA type resource adapters supported on distributed platforms. RRS is the transaction coordinator between WebSphere and other subsystems such as DB2, IMS, CICS on z/OS.

z/OS resource adapters that are capable of using RRS are:

- IMS Connector for Java
- CICS CTG ECI J2EE Connector
- IMS JDBC Connector
- DB2 for z/OS Local JDBC connector when used as a JDBC provider under the WebSphere Relational Resource Adapter (RRA)

Normally, all systems in a Parallel Sysplex share a common set of RRS logs for sync point processing. The log streams can be defined as Coupling Facility log streams or DASD-only log streams (DASD stands for Direct Access Storage Device, a term for a disk drive on zSeries).

When using Coupling Facility log streams, RRS on different systems use shared log streams to keep track of the work. If a system fails, an instance of RRS on another system in the Parallel Sysplex can use the shared logs to take over the work from the failed system.

Note: Use two Coupling Facilities in a Parallel Sysplex to prevent a single point of failure in the sysplex.

8.6.3 XA transactions

WebSphere Application Server V6 enables hot failover of in-transit two-phase commit (2PC) transactions. The XA transaction log for WebSphere on z/OS should be configured to use a shared Hierarchical File System (HFS). Every system in the Parallel Sysplex can be configured to access the shared HFS. Because every server can read another server's transaction log, recovery of an in-doubt transaction is quick and easy. For performance reason, we recommend to use a separate shared HFS for XA transaction logging, rather than using the same HFS that is used for the server configuration.

The log stream can also be used for XA transaction logging. Using Coupling Facility log streams is recommended for sysplex-wide recovery.

The use of a shared HFS for XA transaction logging must be enabled explicitly, Check the Enable high availability for persistent services box in the cluster configuration as shown in Figure 8-7.

Note: Ensure that no ARM policy exists for the cluster.

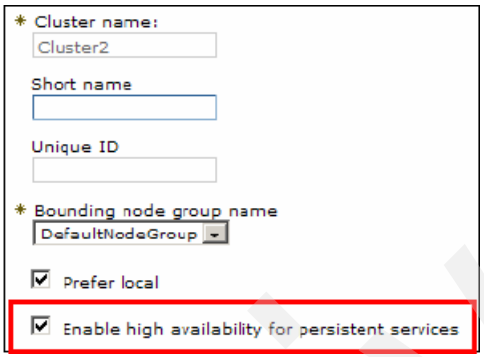


Figure 8-7 Enable HA for persistent services

8.7 HTTP session and stateful session bean failover

This section explains how HTTP session and stateful session bean failover are handled in WebSphere for z/OS.

8.7.1 HTTP session failover

HTTP sessions in WebSphere Application server can be replicated from one server instance to another to prevent single point of failures. This memory-to-memory replication uses the Data Replication Service (DRS) of WebSphere Application Server. Another option for HTTP session state is to persist session data in DB2.

The Web server plug-in keeps a list of the primary and possible backup application servers in the replication domain. When an HTTP session request fails on the primary server, the Web server plug-in sends the request to one of the backup application servers where the session is replicated (or which can retrieve the session from the database).

Application servers on z/OS that are enabled for HTTP session memory-to-memory replication can store replicated HTTP session data in the controller and replicate data to other application servers.

The HTTP session data that is stored in a controller is retrievable by any of the servants of that controller. HTTP session affinity is still associated to a particular servant. However, if that servant should fail, any of the other servants can retrieve the HTTP session data stored in the controller and establish a new affinity.

To handle sessional HTTP requests, the Web server plug-ins for all HTTP servers need to use static IP addresses. In addition, for sessionless HTTP requests, the plug-ins should also use a cell level IP address (DVIPA) so Sysplex Distributor can balance these requests between all systems in the sysplex.

8.7.2 Stateful session bean failover

In WebSphere Application Server on z/OS, the distribution of IIOP requests is handled by the Location Service Daemon. When a request for an EJB lookup arrives at the controller region, the controller region points the requestor to the Location Service Daemon. Then the requestor sends a locate request to the daemon, the daemon assigns a controller to the client, one of the servants of the assigned controller creates the EJB and all subsequent calls to the EJB from the same requestor go through the same controller.

The state replication of a stateful session bean is also based on DRS, similar to HTTP session replication.

The J2EE 1.4 specification requires HTTP session state objects to be able to contain local references to EJBs. Using local references to EJBs has big performance advantages. WebSphere Application Server V6 can collocate stateful session bean replicas and HTTP session replicas for hot failover. In other words, if an HTTP session object in a server contains a local reference to a stateful session EJB instance in the same server, both HTTP session and EJB session are replicated to the same replica in the replication domain. In case of a failover, the HTTP session still holds the local reference to the EJB and is able to locate the local session EJB instance.

Another aspect regarding WebSphere z/OS EJB session failover is failover among servant regions for unmanaged servers. In an unmanaged z/OS server, stateful session bean failover among servants can be enabled. Failover only occurs between the servants of a given unmanaged server. If an unmanaged z/OS server has only one servant, then enabling failover has no effect. To enable failover between servants in an unmanaged server, use the Administrative Console to add a new custom property called EJBContainerEnableUnmanagedServerReplication to the servant JVMs and set it to true.

Because you might not want to enable failover for every single stateful session bean in the EJB container, you can override the EJB container settings at either the application or EJB module level. You can either enable or disable failover at each of these levels. This is the same as WebSphere on other platforms. See 2.6.5, “Stateful session bean failover” on page 73 for more information about this topic.

To enable sysplex-wide distribution of EJB requests, a WebSphere Application Server and its Location Service Daemon must be started on each system. A DVIPA can be used as the sysplex-wide Location Service Daemon IP. The Sysplex Distributor uses this IP to balance requests across the systems. In this configuration, the stateful session EJB sessions must be replicated using DRS. Also, a shared HFS should be used for passivating EJBs.

8.8 JMS failover

WebSphere Application Server V6 introduces a new default messaging provider. See Chapter 12, “Using and optimizing the default messaging provider” of the redbook *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392 for details.

On z/OS, the messaging engine (ME) runs inside a separate address space, called the Control Region Adjunct (CRA). See Figure 8-2 on page 261. A resource adapter in the CRA is responsible for taking a messaging request and sending it to a message dispatcher (RA dispatcher) in a servant region, where the MDB is really dispatched. This resource adapter supports transaction commit and rollback. Messages in MEs can be persisted in a database for recovery.

CRA is a special type of servant region. It is started by Workload Manager for z/OS. When a CRA fails within a server, WLM on z/OS restarts the CRA. Meanwhile the messaging engine in the failed CRA is moved to another server in the cluster and JMS connections are subsequently routed to that ME. Depending on the configuration, the ME is then moved back to the original server's CRA when it is restarted or stays in the CRA. Thus, WLM on z/OS and HAManager complement each other.

8.9 DB2 data sharing

A complete WebSphere high availability solution also requires other components that WebSphere interacts with, to be highly available. These components can be DB2, WebSphere MQ, and so on.

The high availability solution for DB2 on z/OS is DB2 Data Sharing. See Figure 8-8. This function enables multiple applications to read from, and write to, the same DB2 data concurrently. The applications can run on different DB2 subsystems residing on multiple Central Processor Complexes (CPCs) in a Parallel Sysplex.

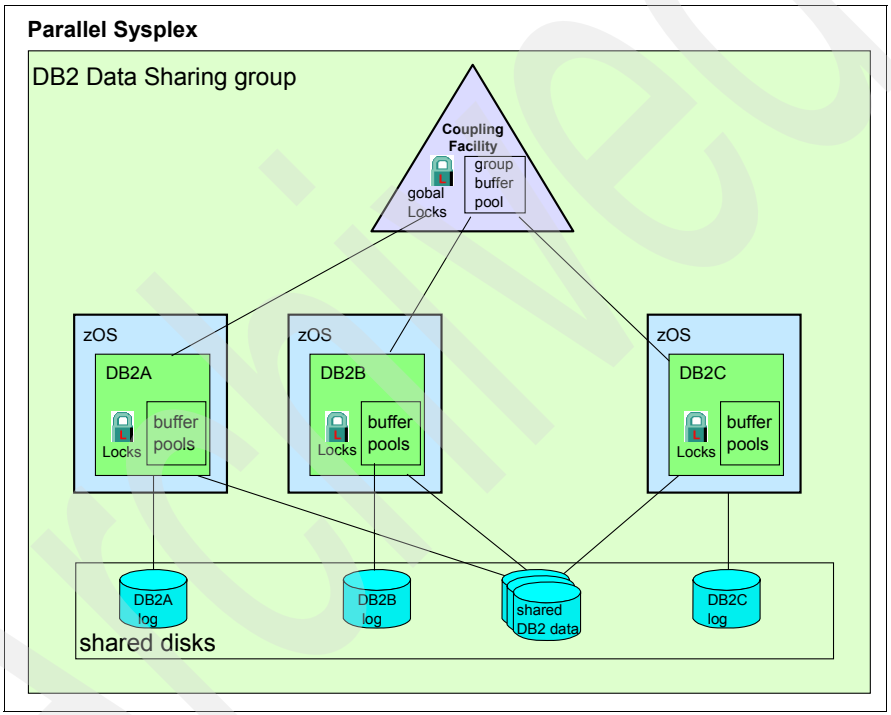


Figure 8-8 A DB2 Data Sharing Group in a sysplex

DB2 data sharing improves the availability of DB2 data, extends the processing capacity of your system, provides more flexible ways to configure your environment, and increases transaction rates. You do not need to change the SQL in your applications to use data sharing, although you might need to do some tuning for optimal performance.

Data sharing provides multiple paths to data, a member can be down, and applications can still access the data through other members of the data sharing

group. When one member is down, Transaction Managers are informed that the member is unavailable, and they can direct new application requests to another member of the group. For more information about DB2 for z/OS Data Sharing, see the DB2 InfoCenter at:

http://publib.boulder.ibm.com/infocenter/dzichelp/index.jsp

8.10 WebSphere MQ for z/OS high availability

In addition to using MQ clusters as a high availability choice (see 15.5, “WebSphere MQ (and other messaging providers)” on page 579), WebSphere MQ on z/OS can take advantage of MQ queue sharing. See Figure 8-9 on page 279. A queue sharing group is a collection of queue managers on different systems in a Parallel Sysplex that have access to the same set of shared queues and to shared definitions. The WebSphere MQ queue sharing group configuration requires DB2 data sharing.

The shared WebSphere MQ object definitions are stored in DB2 tables, and messages belonging to a shared queue reside in the Coupling Facility. Two Coupling Facilities can be used in a Parallel Sysplex to prevent a single point of failure.

For more details on WebSphere MQ, see *WebSphere MQ in a z/OS Parallel Sysplex Environment*, SG24-6864.

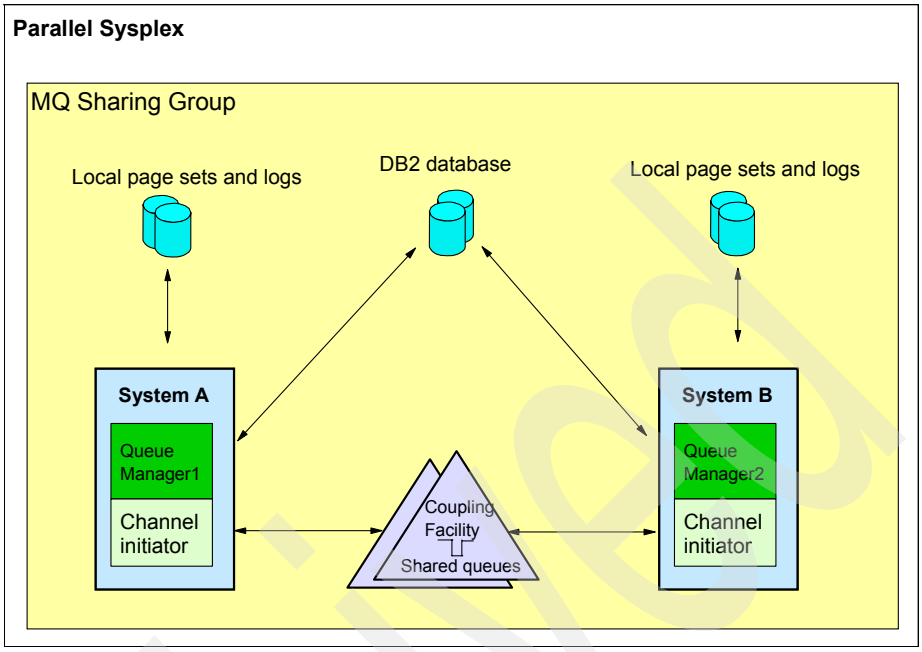


Figure 8-9 WebSphere MQ queue sharing

8.11 A sample high availability configuration

After having explained some basic concepts of WebSphere high availability on z/OS, let us put together a sample high availability configuration for z/OS. See Figure 8-10.

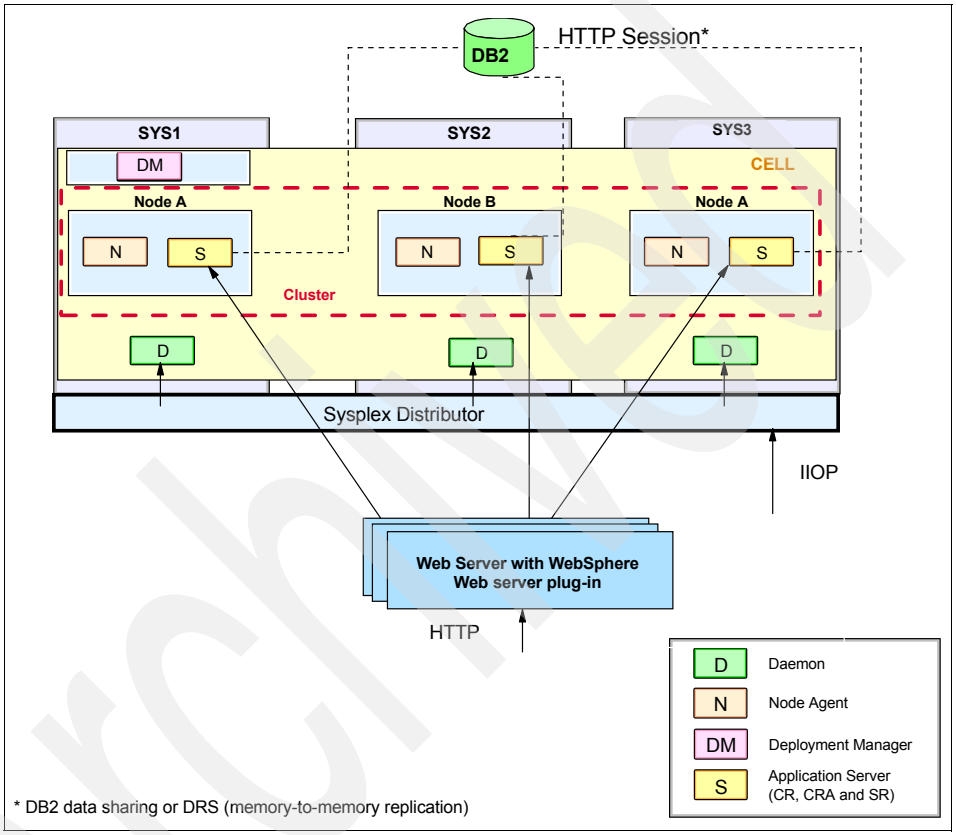


Figure 8-10 High availability WebSphere in a sysplex configuration,

This high availability WebSphere configuration on z/OS requires a Parallel Sysplex environment plus the following setups within and outside the Parallel Sysplex.

Setup outside the Parallel Sysplex includes:

- Redundant network path leading to the Web servers and application servers.
- Redundant Web servers. In many cases, HTTP servers are on another platform instead of z/OS while application servers run on z/OS.

- A WebSphere Web server plug-in must be installed in each of the Web servers and configured to use HTTP DVIPA for sessionless requests and static IP addresses for sessional requests.

Setup within the Parallel Sysplex includes:

- A highly available Parallel Sysplex with two Coupling Facilities and at least three LPARs hosting application servers within a cell. The LPARs in the sysplex should be on separate hardware instances to prevent a single point of failure. A high availability configuration without the need to perform software, hardware or application updates only needs two LPARS. We recommend a third LPAR in the configuration such that the configuration is highly available even when one of the LPARs is brought down for upgrade.
- A WebSphere Application Server for z/OS node on each LPAR that is configured into a Network Deployment cell. The Deployment Manager server (required, and configured on its own node) can be configured on one of the LPARs or on a separate LPAR. Also note that there is a daemon process (WebSphere CORBA Location Service) on each LPAR that has one or more nodes in the same cell.
- An application server defined on each node, and formed into a server cluster which consists of at least three nodes from three LPARs.
- A Dynamic virtual IP address (DVIPA) defined through the z/OS Sysplex Distributor as the daemon IP name for the cell. This IP address enables WLM-balanced routing and failover between the LPARs for IIOP requests.
- A Dynamic virtual IP address (DVIPA) defined through Sysplex Distributor as the HTTP transport name for the cell. This IP address enables WLM-balanced routing and failover between the LPARs for sessionless HTTP requests.
- A static IP address is required for each node as an auxiliary HTTP transport name for the cell. This enables directed HTTP routing for sessional HTTP requests.
- If using HTTP sessions, the session state must be shared between the cluster member using DRS or session data must be stored in DB2. If you are using stateful session Enterprise JavaBeans (EJBs), the stateful session must be replicated using DRS, and a shared HFS must be configured for passivating session EJBs. Note that using stateful session beans is not a best practice.

8.12 Hardware, software, and application upgrade

Certain hardware and software upgrades and installations can be performed in the Parallel Sysplex in a nondisruptive manner, such as varying a processor online or offline. Certain other software and hardware changes require a system to be taken offline. For a more detailed discussion of this topic see *WebSphere for z/OS V6 High Availability*, SG24-6850.

Application updates in WebSphere Application Server on z/OS are performed the same way as in WebSphere V6 on other platforms. See Chapter 5, “High availability application administration” on page 141 for more information.

8.13 WebSphere Application Server for Linux on zSeries

WebSphere Application Server for Linux on zSeries works the same way as WebSphere on any other Linux platform. High availability can be achieved using the WebSphere provided facilities and clustering products, such as Tivoli System Automation for Multiplatforms. See Chapter 10, “WebSphere and IBM Tivoli System Automation” on page 367 for more information.

8.14 Reference

The following resources are available:

- *Parallel Sysplex Cluster Technology* at:

  `http://www.ibm.com/servers/eserver/zseries/pso/sysover.html`

- *Leveraging z/OS TCP/IP Dynamic VIPAs and Sysplex Distributor for higher availability* at:

  `http://www.ibm.com/servers/eserver/zseries/library/techpapers/gm130165.html`

- DB2 Information Management Software Information Center for z/OS Solutions at:

  `http://publib.boulder.ibm.com/infocenter/dzichelp/index.jsp`

- *WebSphere for z/OS V6 High Availability*, SG24-6850
- *WebSphere MQ in a z/OS Parallel Sysplex Environment*, SG24-6864

Part 5

Using external clustering software

9

Configuring WebSphere Application Server for external clustering software

This chapter provides information about how to configure WebSphere Application Server to benefit from external clustering software. We explain how to make the Deployment Manager or Node Agent and application server highly available in this chapter.

Also covered in this chapter is a typical topology, consisting of several application servers connected to some backend services. Those services, for example database, messaging, or LDAP services, reside on clustered remote systems.

This discussion is independent from the chosen clustering software. A more detailed perspective about how to configure WebSphere V6 to depend on a special clustering software is discussed in the following chapters:

- Chapter 11, “WebSphere and IBM HACMP” on page 417
- Chapter 10, “WebSphere and IBM Tivoli System Automation” on page 367
- Chapter 12, “WebSphere and VERITAS Cluster Server” on page 445
- Chapter 13, “WebSphere and Sun Cluster” on page 483

9.1 Introduction

The high availability capabilities of WebSphere V6 have been further improved compared to previous versions. WebSphere Application Server V6 can be used as part of an overall 99.999% availability solution. However, to achieve this, your environment has to be highly available. For that reason, you also need to consider the surrounding services. We cover some WebSphere V6 settings for interacting with typical enterprise services.

You should modify some parameters in the WebSphere V6 configuration to leverage the benefits of using external clustering software in your environment. These settings are driven by the systems architecture and thus valid for all flavours of external clustering software. We discuss the settings for:

- Node Agent and application server HA
- Deployment Manager HA
- Configuring JMS resources
- Configuring JDBC/data sets
- Configuring security

Platform-clustering software packages can be used to enhance the availability of the entire WebSphere system. The unit of failover usually includes a collection of network definitions and disk storage, and one or more services such as the DB2 database server (or other database servers), the HTTP server, firewalls, LDAP server, WebSphere Application Server, WebSphere Node Agent or Deployment Manager. However, it is not standardized, and different vendors use different terms. For example, the unit of failover in IBM HACMP or HACMP/ES Cluster is called an application server, while the unit of failover in IBM Tivoli System Automation (TSA) is called resource group. Sun Cluster calls its failover unit a logical host, and the unit of failover in VERITAS Cluster Server is called Service Group.

9.1.1 IP-based cluster failover versus non-IP based cluster failover

When people talk about clustering, they often mean different things. The reason is that there are two kinds of cluster failovers:

- IP-based cluster failover:

  This approach deals with a Virtual IP Address or IP Alias. The IP Alias is only used for one system at a time. In case of a failover, the IP Alias is moved to the other system. All applications or services have to use the Virtual IP Address to access the cluster.

  Examples for software using this approach are IBM HACMP, TSA, Sun Cluster, VERITAS Cluster Server.

- Non-IP cluster failover:

  In this case, there is a process failover, regardless of the TCP/IP connection parameter. Such a failover is provided by an application or a vendor specific protocol.

  For example, WebSphere workload management (WLM) and the WebSphere HAManager use this approach.

Usually, IP-based cluster failover is slower (one to five minutes), and non-IP cluster failover is very fast (instantaneous). While the WebSphere V6 HAManager does not require extra software, most non-IP cluster failover still relies on external cluster software to provide the cluster information.

9.1.2 High availability configuration types

There are two types of highly available data services:

- A scalable data service spreads an application across multiple nodes to create a single, logical service. Scalable services leverage the number of nodes and processors in the entire cluster on which they run. In case of a failure this solution means a loss in capacity.
- A failover data service runs an application on only one primary node in the cluster at a time. Other nodes might run other applications, but each application runs on only a single node. If a primary node fails, the applications running on the failed node failover to another node and continue running. We can further divide this category into two subcategories:
  - Active/Active mode, where two services reside in two nodes that are configured as mutual failover.
  - Active/Passive mode, where one node is configured as the primary to run the service, while the other node is configured as a hot standby.

  The configuration for both modes is very similar. The advantage of the Active/Active mode configuration is lower hardware cost. However, the service performance is reduced when a failover occurs. The advantage of the Active/Passive mode configuration is steady performance, but redundant hardware is needed. Furthermore, the Active/Active mode configuration might have twice as many interruptions as the Active/Passive mode configuration, because a failure in any of two nodes can cause a failover.

9.1.3 Failover terms and mechanisms

An object or an application includes two distinct aspects: functions and data. Therefore, we have process availability and data availability. If a function is not associated with individual data or states, it is easy to achieve high availability by simply restarting this function process when the old process crashes. However, the reality is that functions are associated with individual data or state, some with persisted data in database or files, such as Entity EJBs. We need to make data management systems highly available to all processes and ensure data integrity because the failed process might damage data.

- Failover

  Failover refers to the single process that moves from the primary system to the backup system in the cluster. The failure recovery includes several steps:

  a. Stop and exit the failed process.
  b. Release the resources.
  c. Detach the disk array.
  d. Reattach the disk array to the backup system.
  e. Check the disk and file system.
  f. Repair the data integrity.
  g. Gain all resources for running the process in the backup system.
  h. Start the process in the backup system.

  This failover process takes several minutes after the fault is detected. This approach can be used for both function-centric or data-centric applications for both Active/Passive and Active/Active configurations.

- Fail back, fallback

  Fail back or *fallback* is similar to failover, but occurs from the backup system to the primary system when the primary system is back online. For mutual takeover, because the backup node has its original application running, as shown in Figure 9-8 on page 297, failing back improves the performance of both applications.

9.2 Standard practice

Every project starts with a planning phase. In this section, we provide some tasks, that are, from our point of view, just a starting point. For more information about planning, we recommend the *IBM WebSphere V6 Planning and Design Handbook*, SG24-6446 and *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392. In addition, your local IBM representative can help you.

9.2.1 Gathering non-functional requirements

To make a decision on how to set up your systems, you have to gather some relevant information. Some of the most important information are the non-functional requirements. Depending on those non-functional requirements, your system architecture, or operational model, should be designed. This is usually a high-level design.

The non-functional requirements are determined through your business needs. They directly impact the cost of your solution, but they also define the gainable service level.

If you come to the conclusion that you need to set up WebSphere Application Server using some kind of platform-specific clustering software, then there are different ways to do this as described in the following sections.

For more information about non-functional requirements, see the article *Quality busters: Forget the environment* at:

http://www.ibm.com/developerworks/web/library/wa-qualbust1/index.html

9.2.2 Choosing the HA configuration type

As discussed in “High availability configuration types” on page 287, there are different ways on how to configure your high availability solution. This section gives a short overview of the failover behavior of the Active/Passive versus the Active/Active configuration type to help you choose the right configuration type for your environment.

Configuration in Active/Passive mode

During normal operations only one system runs the process. The process is down on the second system. Both systems share the same file system. The following figures show Active/Passive failover configurations. Figure 9-1 shows the configuration before the failover.

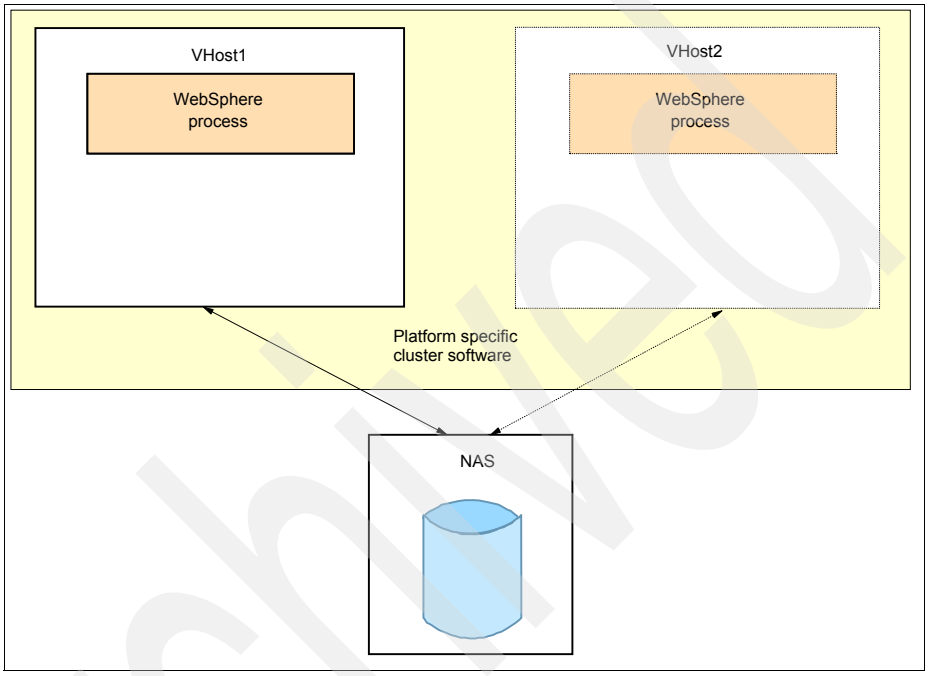


Figure 9-1 Active/Passive configuration before failover

In case of a failure on the active system, the clustering software detects that failure and initiates a failover. Be aware that there is a downtime at the time of the failure! This is because the running process has failed and the backup process waits to get started by the cluster software. Figure 9-2 shows this situation.

Important: Make sure that your failure detection mechanism is set up properly. How to set up the monitoring varies between the different cluster software products.

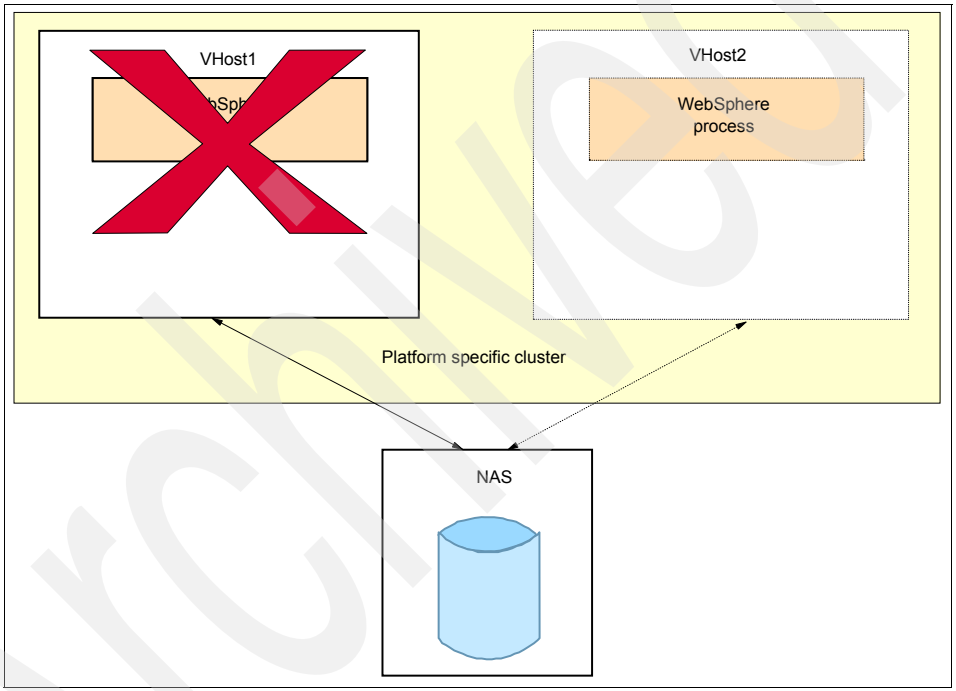


Figure 9-2 Active/Passive configuration at failure time

The failover starts after failure detection. For a WebSphere Application Server process, a typical failover means that the following occurs (as shown in Figure 9-3):

1. Cleaning up the process.
2. Releasing the disk.
3. Shutting down the interface to which the Virtual IP Address (VIP) is bound.
4. Mounting the disk on the backup system.
5. Checking the disk.
6. Starting the interface to which the VIP is bound.
7. Starting the WebSphere process.

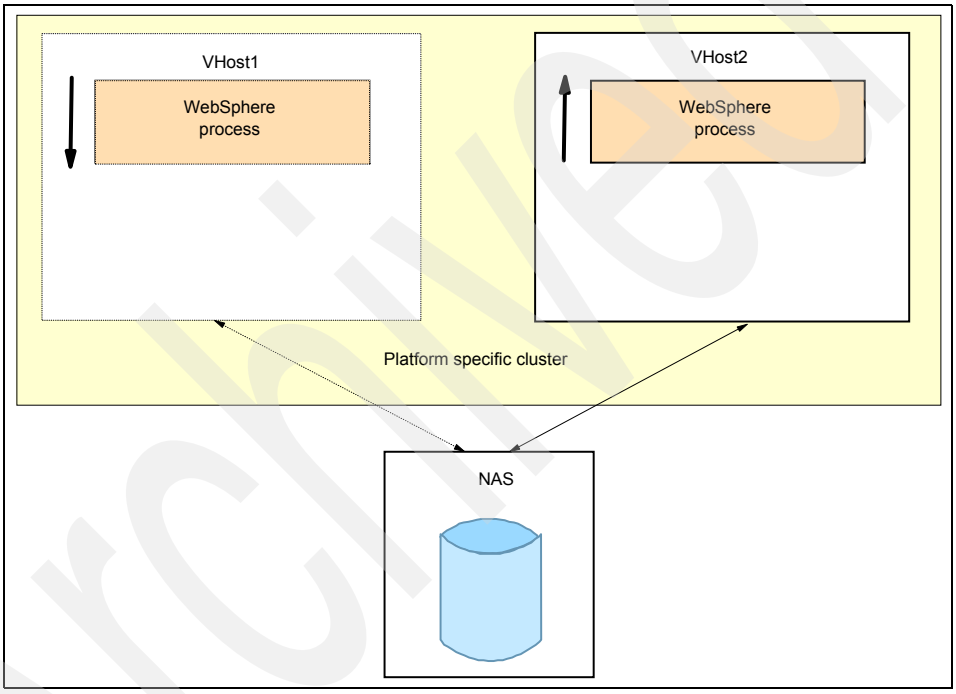


Figure 9-3 Active/Passive configuration during failover

After the failover has finished, the process runs on the backup system as illustrated in Figure 9-4.

Important: Cluster software does not avoid failures! It however helps to determine the maximum downtime if a process fails. The amount of process downtime is *time of failure detection + time to failover*.

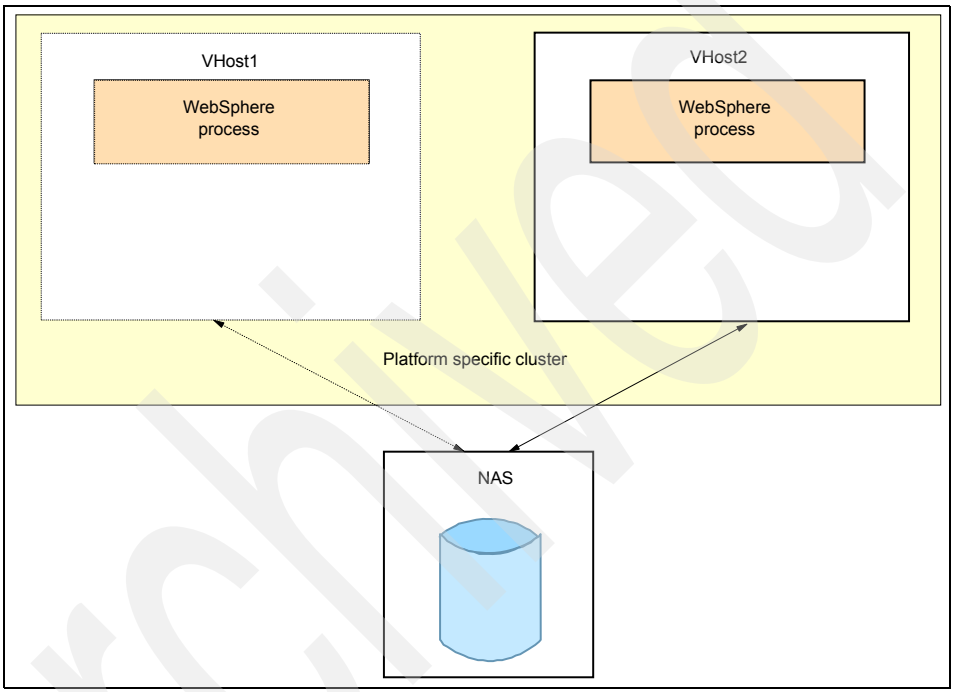


Figure 9-4 Active/Passive configuration after failover

Configuration in Active/Active mode

You can spread multiple *independent* applications across your cluster servers. This is called an Active/Active setup. Such a configuration allows for a better use of your hardware because all systems are constantly in use. In our example in Figure 9-5, there is a WebSphere process1 on a system called VHost1 and a second WebSphere process2 on a system called VHost2.

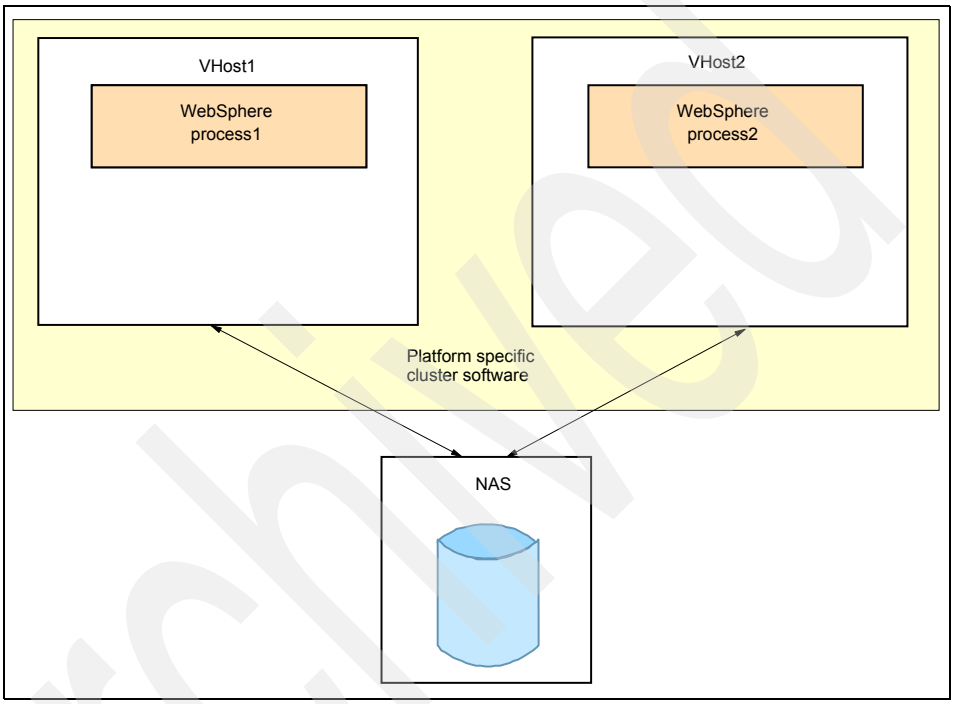


Figure 9-5 Active/Active configuration before failover

In case of a failure on, for example, VHost1, the clustering software detects that failure and initiates a failover. There is also a downtime for the failed process! This is because the backup process waits to get started by the cluster software. Figure 9-6 shows this situation.

Important: Make sure that your failure detection mechanism is set up properly. How to set up the monitoring varies between the different cluster software products.

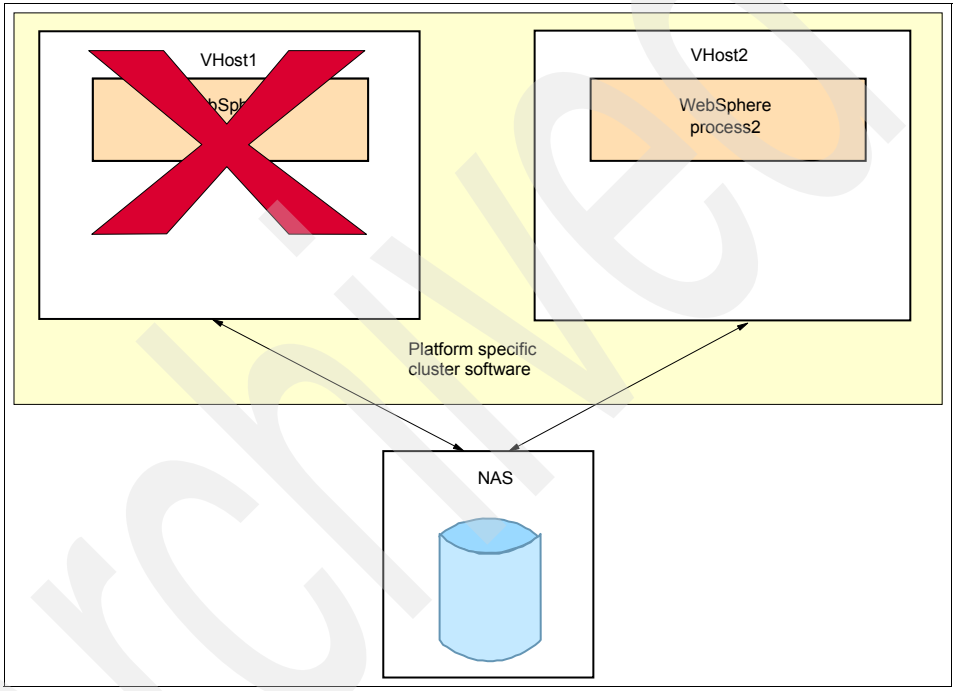


Figure 9-6 Active/Active configuration at failure time

As soon as the failure is detected by the cluster software, the failover starts. For a WebSphere Application Server process a typical failover means that the following occurs (as shown in Figure 9-7):

1. Cleaning up the process.
2. Releasing the disk.
3. Shutting down the interface to which VIP (for this particular process!) is bound.
4. Mounting the disk on the backup system.
5. Checking the disk.
6. Starting the interface to which the VIP for this particular process is bound.
7. Starting the WebSphere V6 process.

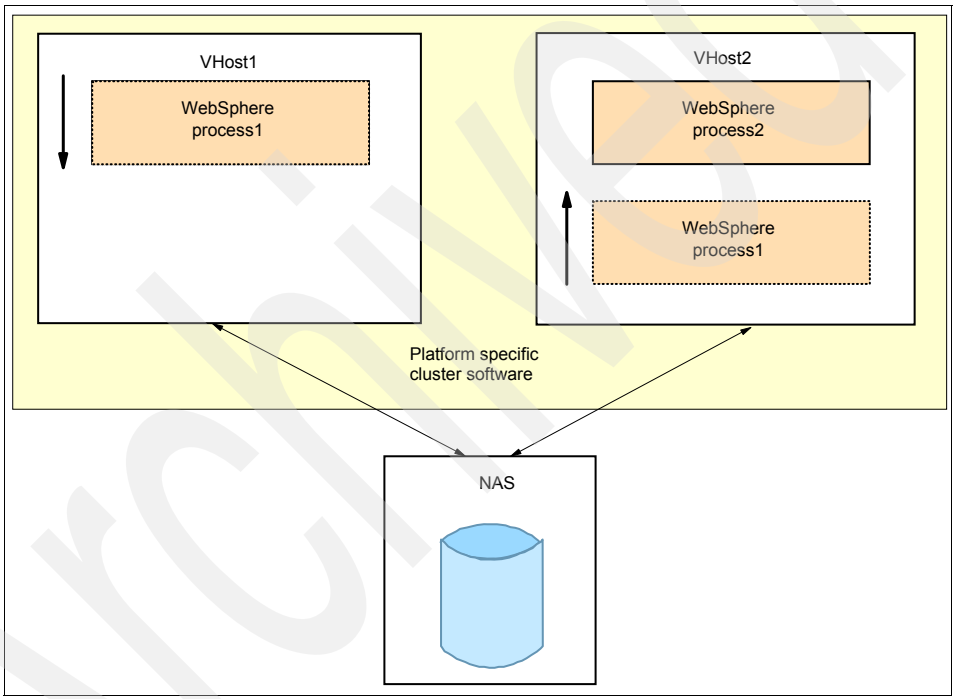


Figure 9-7 Active/Active configuration during failover

After the failover has finished, the process runs on the backup system in addition to the other processes running on that system as shown in Figure 9-8. Which means you now have reduced system capacities for all processes.

Important: Cluster software does not avoid failures! It however helps to determine the maximum downtime if a process fails. Again, a process failure causes a maximum downtime of *time of failure detection + time to failover*. Additionally, a process failure causes more load on the backup system in an Active/Active configuration.

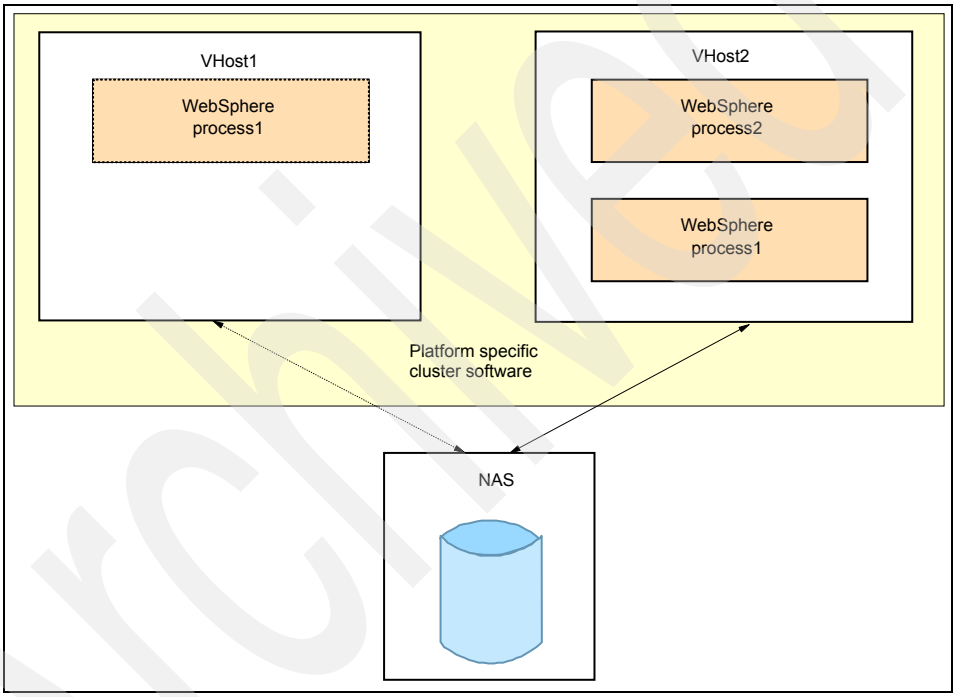


Figure 9-8 Active/Active configuration after failover

9.2.3 Configuring the environment: WebSphere Application Server binaries and profiles

WebSphere Application Server V6 separates read only binary data from customizable configuration data. The installation procedure provides the binary data while the profile creation after installation provides the configuration data. This approach allows you to set up independent instances of WebSphere Application Server that share one set of binaries. You need to decide where to

put the binaries and the profile data in your HA environment. There are two options:

- Everything on shared disks
- Only the profiles on shared disks

See the sections titled *Preparing* of each scenario in the sections that follow for more information about this topic. The “Installing WebSphere Application Server Network Deployment” sections of each scenario contain instructions on how to create the proper profiles.

For more information about profiles, see Chapter 4 of *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

9.2.4 Testing

After you finished the installation and configuration, it is necessary to test the failover abilities of your environment. Possible test scenarios are:

- Simulating a network failure
- Shutting down the system that is currently active
- Deleting the pid file of your application server (if this file is monitored)
- Killing the process of your application server
- And more ....

These tests should be extended by additional ones - depending on your needs. This list does not cover all possible test scenarios.

9.3 Deployment Manager high availability

Application errors occur more often than system failures. The first option to increase process or application availability is therefore to try a local restart of the failed application. If successful, this approach minimizes downtime as a process restart is significantly faster than a failover to a remote system. See 3.4, “Restarting WebSphere processes as an OS service” on page 117 for information about how to do this.

In fact, there is no need for a highly available Deployment Manager in WebSphere V6. All critical singleton services can now run on different processes in the cell and are controlled by the HAManager. Therefore, the Deployment Manager is no longer a single point of failure. See Chapter 3, “WebSphere administrative process failures” on page 103 for more information.

If you decide however, that you still want to make your Deployment Manager highly available, then the information provided in this section should be helpful.

9.3.1 Preparing

Consider the following regarding your WebSphere installation:

- Make sure you choose the installation method that fits into your systems architecture. Basically there are two different ways to install a Deployment Manager (DMgr) on a platform-specific cluster:

  a. Installing the binaries on a shared disk that is attached to all cluster members.

  b. Installing the binaries locally on every cluster member and placing only the profiles on a shared disk.

  Note: The term *cluster member* in this context refers to the platform-specific cluster, not to a WebSphere cluster.

  Both methods are applicable, but if disk space is not an issue, we recommend to use the second option.

- To install WebSphere V6, you have to:

  a. Install the binaries.
  b. Create a profile.

  Depending on which installation method you choose, the binaries and the profile are on the same disk or mountpoint (in case of a volume), or they are separated.

- Installing IBM WebSphere Application Server Network Deployment V6 on an platform-specific cluster that uses the IP-based cluster failover mechanism demands that your host name is bound to your IP Alias (Virtual IP Address). This IP Alias is used to access your system.

  When creating a profile, you are prompted for the host name. To benefit from the failover technique of any platform-specific clustering software, you *must* use the host name related to your IP Alias.

  To check the DNS mapping using a command line, enter this command:

  ```
  $ nslookup your_ip_address
  ```

Redbook sample scenario for Deployment Manager HA

Figure 9-9 shows our lab setup. We used two systems in an Active/Passive configuration which means that the Deployment Manager is running only on one of the systems at a time. We decided to install the WebSphere V6 binaries locally on both systems for the VERITAS Cluster Server scenario while installing them on the shared disk for the other cluster software scenarios. In the first case, only the Deployment Manager profile resides on the NAS and is accessible by both systems.

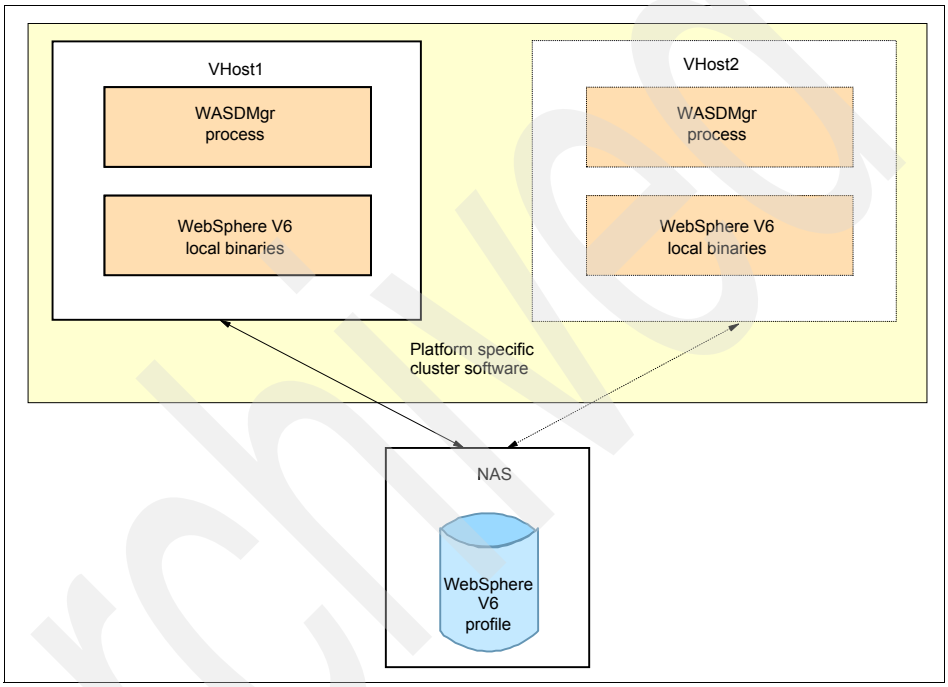


Figure 9-9 Lab example showing two systems connected to NAS for DMgr HA

Important: The value of your *WAS_HOME* variable has to be the same on both systems!

9.3.2 Installing WebSphere Application Server Network Deployment

To install IBM WebSphere Application Server Network Deployment V6, follow the instructions that are provided in the document *IBM WebSphere Application Server Network Deployment V6: Installing your application serving environment* at:

ftp://ftp.software.ibm.com/software/webserver/appserv/library/v60/wasv60Ond_gs.pdf

Make sure that you select the correct location for the binaries, depending on your decision for local or shared disks. After the installation of the code has completed, you need to create a Deployment Manager profile. WebSphere Application Server V6 provides two ways to create profiles:

- Using the Profile creation wizard
- Using the **wasprofile** script on a command line

Example 9-1 shows how to create a Deployment Manager profile using the command line.

Example 9-1 Creating DMgr profile using the command line

```
$ ./wasprofile.sh -create -profileName your_profilename -profilePath
/NAS_Path/path_of_your_profile -templatePath
/IBM/WebSphere/AppServer/profileTemplates/dmgr -nodeName your_profile_node
-cellName your_cell_name -hostName your_virtual_hostname
```

You can also use the Profile creation wizard to create a Deployment Manager profile. To start the wizard, for example on Linux, use this command:

```
<WAS_HOME>/bin/ProfileCreator/pctLinux.bin
```

Then choose the appropriate profile type as shown in Figure 9-10.

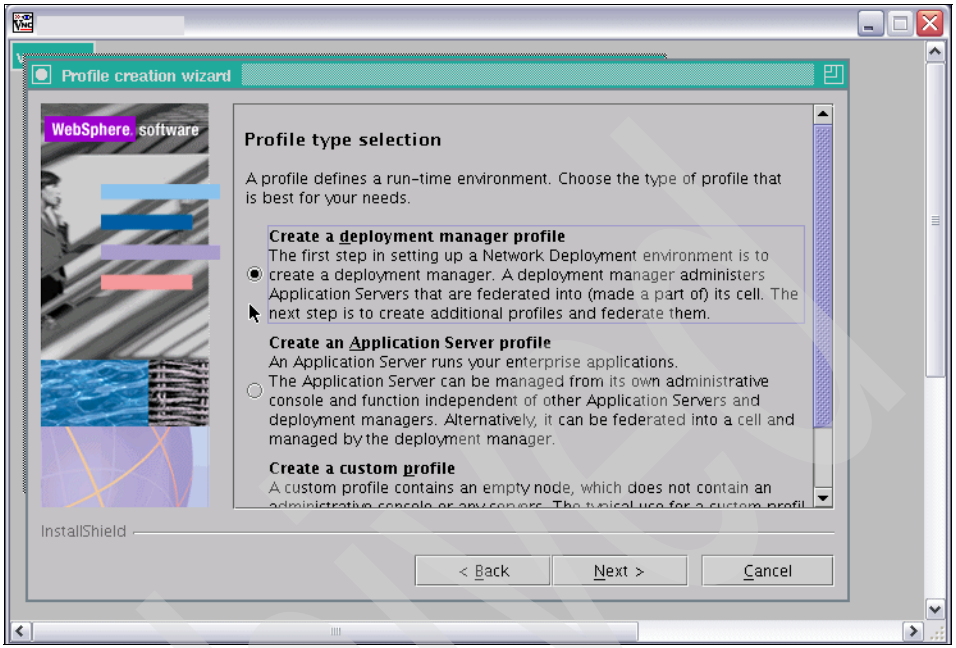


Figure 9-10 Profile creation wizard for a Deployment Manager profile

For our lab example, the DNS entry of the IP Alias is VCluster.ibmredbook.com. This value must be entered as the Host name, which is shown in Figure 9-11.

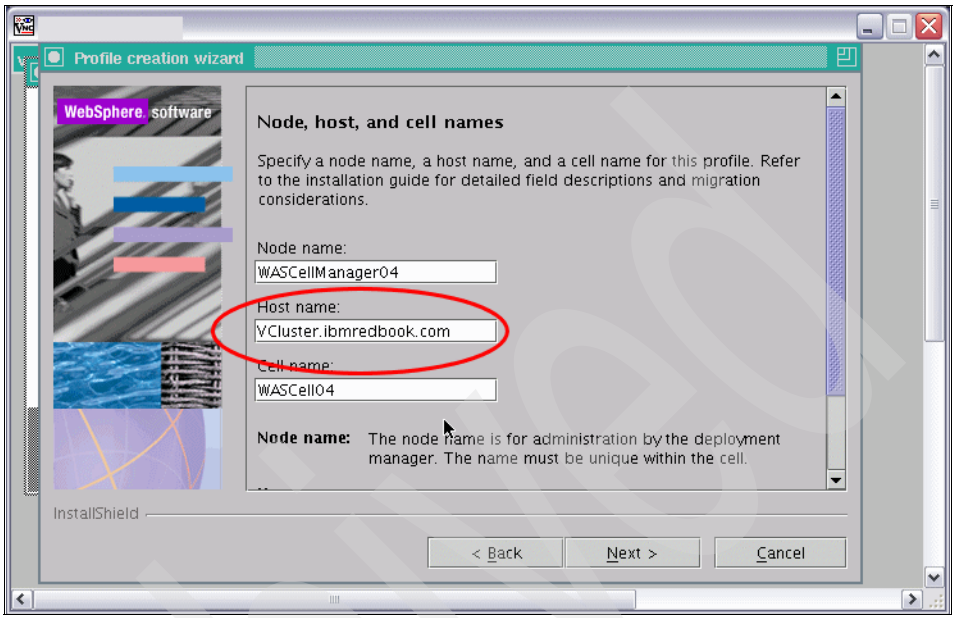


Figure 9-11 Profile creation wizard, prompting for the Host name, DMgr profile

9.3.3 Configuring the clustering software

As mentioned before, the first and easiest option to enhance Deployment Manager availability is to monitor the process using the operating systems's capabilities. See 3.4, "Restarting WebSphere processes as an OS service" on page 117 for information about how to do this.

The configuration steps to make WebSphere Application Server highly available using a platform-specific clustering technique vary from product to product. But you always have to define some kind of *failover unit*. These units are switched between the nodes in case of a failover. Usually the failover units need a start, stop, and monitor program. You can find detailed instructions on how to set up HACMP, Sun Cluster, Tivoli System Automation, and VERITAS Cluster Server in the appropriate chapters of Part 5.

Independent from the clustering software you choose, do these steps during a failover:

- Stop the Deployment Manager process in the failed host
- Release the shared disk and other resources from this node
- Remove the service IP address from the failed node
- Mount the shared disk to the other, healthy node
- Add the same service IP address to the adopted node
- Start the Deployment Manager process

9.4 Node Agent and application server high availability

Although it is possible to make the Node Agent highly available through third-party clustering software, we do not recommend that you do so. Often a simple restart of the process solves your problem.

Therefore, before failing over the Node Agent process to another system, you should try to restart it on the local system. If you are able to restart the process locally, the downtime is greatly reduced compared with failing over to the backup system. See 3.4, “Restarting WebSphere processes as an OS service” on page 117 for information about how to restart the process automatically.

If you decide that you want to make your Node Agent highly available using clustering software, this section describes what you need to consider.

9.4.1 Preparing

Basically, the same considerations as for Deployment Manager high availability also apply for Node Agent and application server HA:

- As described in “Deployment Manager high availability” on page 298, specifically in the Preparing section, there are two different ways to install WebSphere on a platform specific cluster:

  a. Installing the binaries on shared disks that are accessible by all cluster members.

  b. Installing the binaries locally on every cluster member and placing only the profiles onto the shared disks.

  Even though either method is applicable, we recommend to use the second option as long as there are no disk space issues.

- To install WebSphere V6, you have to:

  a. Install the binaries.
  b. Create a profile.

Depending on the chosen installation method, the binaries and the profile are located on the same disk or mountpoint (in case of a volume), or separated.

- On a platform-specific, IP-failover based cluster, make sure that your host name is bound to your IP Alias (Virtual IP Address). You have to provide the DNS name that is mapped to that IP Alias as the Host name during profile creation.

Example 9-2 Checking the DNS mapping of the IP Alias

```
$ nslookup your_ip_address
```

A redbook sample scenario for Node Agent and application server HA

Figure 9-12 on page 306 shows our lab setup. We used two systems in Active/Passive configuration. As a result of that, there is only one system running the Node Agent or application server at a time. As described for the Deployment Manager, we split up our installation. The binaries are located on local disks and the profiles are placed on the shared disk drive. In our case, the shared disk is an NAS that can be concurrently accessed by both systems.

As for the Deployment Manager, we decided to install the WebSphere V6 binaries locally on both systems for the VERITAS Cluster Server scenario while installing them on the shared disk for the other cluster software scenarios. In the first case, only the application server profiles reside on the NAS and are accessible by both systems.

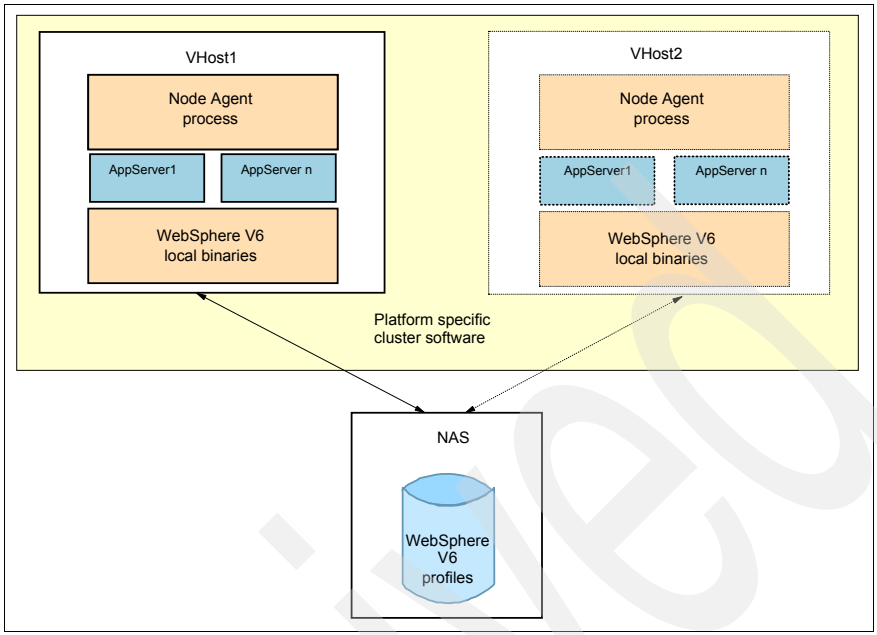


Figure 9-12 Lab example showing two systems connected to NAS for Node Agent HA

Important: The value of your *WAS_HOME* variable has to be the same on both systems!

9.4.2 Installing WebSphere Application Server Network Deployment

First you need to install the Network Deployment software. Choose the correct location for the binaries (shared or local disk). For more information about installation see the document *IBM WebSphere Application Server Network Deployment V6: Installing your application serving environment*, which can be found at:

```
ftp://ftp.software.ibm.com/software/webserver/appserv/library/v60/
wasv600nd_gs.pdf
```

When the code is installed, the next step is to create the profile. To create a managed node (or Node Agent), you need to create a Custom profile or Application Server profile and federate it into a cell. To create a profile:

- Use the Profile creation wizard.
- Use the **`wasprofile`** script on a command line.

In our example, we created the profile using the command line, as shown in Example 9-3.

Example 9-3 Creating a custom profile using the command line

```
$ ./wasprofile.sh -create -profileName your_profilename -profilePath
/NAS_Path/path_of_your_profile -templatePath
/IBM/WebSphere/AppServer/profileTemplates/managed -nodeName your_profile_node
-cellName your_cell_name -hostName your_virtual_hostname
```

You can also use the Profile creation wizard to create a profile of a managed node. To start the wizard, for example on Linux, use the following command:

```
WAS_HOME/bin/ProfileCreator/pctLinux.bin
```

Then choose the appropriate profile type. In our case, the Application Server profile was chosen, as shown in Figure 9-12 on page 306.

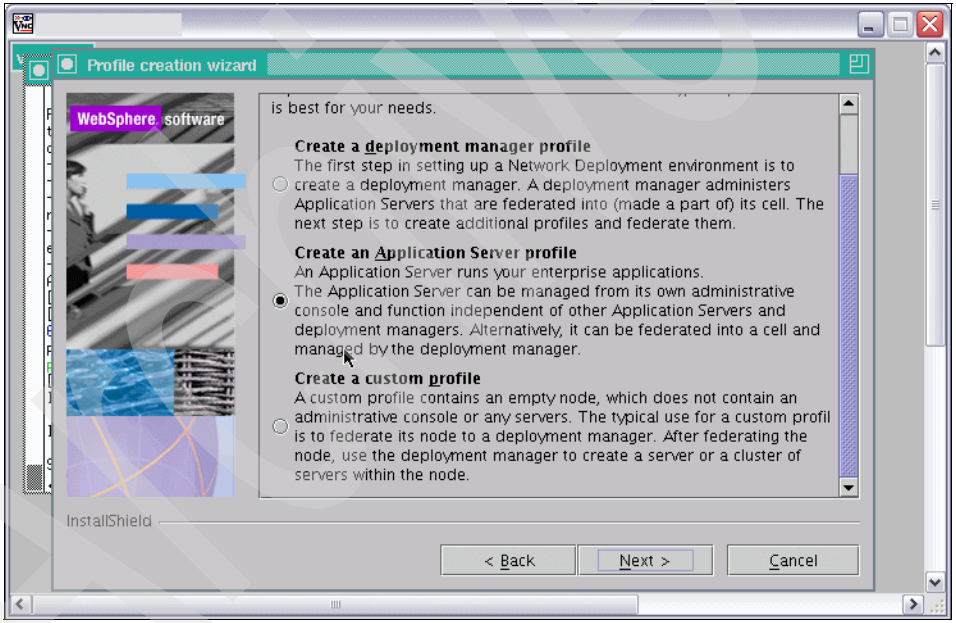


Figure 9-13 Choosing the Application Server profile type - Application Server profile

Attention: To federate an Application Server profile into a cell you have to run the **addNode** command manually after profile creation. You can also federate a profile into a cell *during* profile creation. In this case, you need to create a custom profile.

In our example, the DNS entry of the IP Alias is VCluster.ibmredbook.com. This name must be entered as the Host name during profile creation, as shown in Figure 9-14.

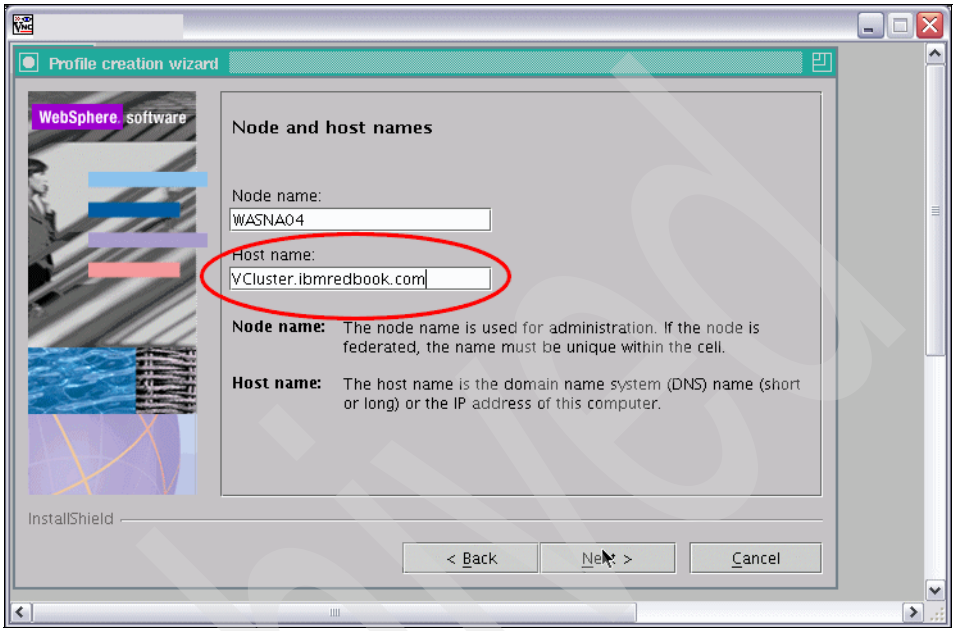


Figure 9-14 Profile creation wizard, prompting for the host name

9.4.3 Configuring the clustering software

The first option to enhance the Node Agent's availability is to monitor the process using the operating system' capabilities. See 3.4, "Restarting WebSphere processes as an OS service" on page 117.

The configuration steps to make WebSphere Application Server highly available using a platform-specific clustering technique vary from product to product. But you always have to define some kind of *failover unit*. These units are switched between the nodes in case of a failover. Usually the failover units need a start, stop, and monitor program. You find detailed instructions on how to set up HACMP, Sun Cluster, Tivoli System Automation, and VERITAS Cluster Server in the appropriate chapters of this part.

9.5 Common advanced topology

Because your environment consists of more than just the WebSphere V6 components, you have to consider also other services and their availability.

We found that the scenario shown in Figure 9-15 is a typical topology. To provide high availability, there are some services outside of WebSphere Application Server, located on a platform-specific cluster. In this scenario it is the Database, LDAP, and IBM WebSphere MQ.

Even though the Deployment Manager does not necessarily have to be highly available, we placed it on the existing cluster. As mentioned in 9.3, “Deployment Manager high availability” on page 298, this is nice to have but not a must.

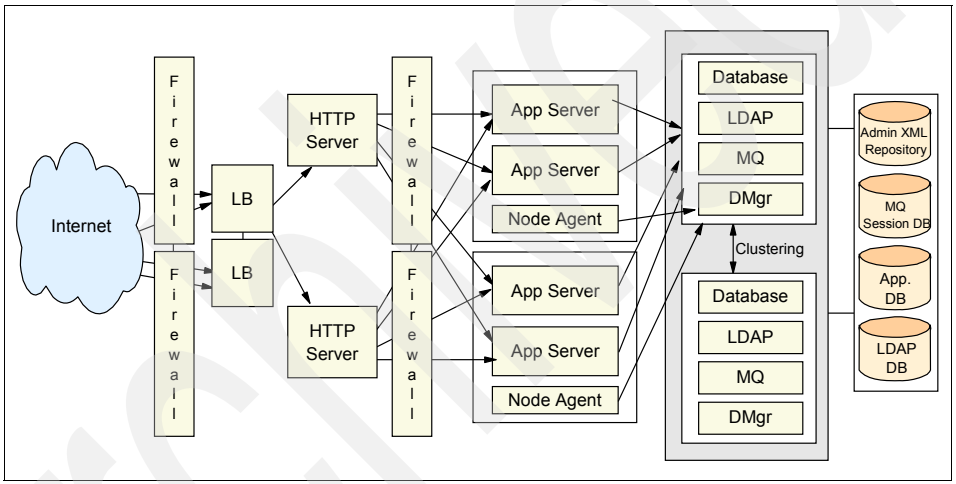


Figure 9-15 Typical example of a real life environment

In such an environment, you have to be aware of some general configuration topics. As you connect to remote systems, there are some settings, regarding time-outs or retry intervals, that have to be set correctly. These settings are covered in this section.

9.5.1 Connecting to a remote database

For general information about how to configure WebSphere V6 resources, especially JDBC resources, see *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

In case the database you are using resides on a platform-specific cluster, you have to be aware of the possible service interruption during a failover. As mentioned in 9.2.2, “Choosing the HA configuration type” on page 289, the maximum downtime of a process, in this case the database, consists of *the time to detect the failure + the time to failover*. It is important to identify these two times and to consider them when setting your datasource connection pool properties. There are two important settings:

- Connection timeout
- Server name

Connection timeout

The connection timeout defines the maximum wait period for a new connection request before it throws a ConnectionWaitTimeoutException. This means, for example, that if your maximum downtime (time to detect a failure + time to failover) is 300 seconds, but your connection timeout is set to 180 seconds, this might cause unnecessary errors.

To verify/change your Connection timeout setting, go to **Resources** → **JDBC providers** → ***JDBC_provider_name*** → **Data sources** → ***Data_source_name*** → **Connection pool properties** in the Administrative Console. See Figure 9-16.

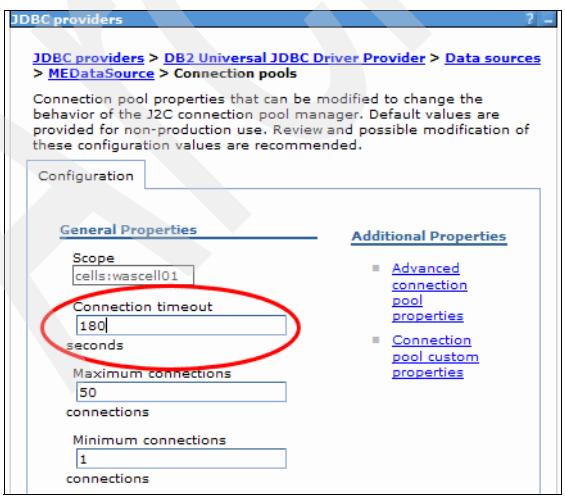


Figure 9-16 Setting the connection timeout

Server name

The Server name for the database on the cluster must be the IP Alias, as configured in the DNS. In our example the IP Alias of the cluster is mapped to VCluster.ibmredbook.com.

To change or verify the Server name in the Administrative Console, go to **Resources** → **JDBC providers** → ***JDBC_provider_name*** → **Data sources** → ***Data_source_name*** as shown in Figure 9-17.

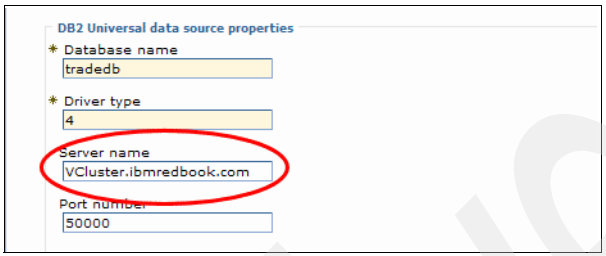


Figure 9-17 Setting the Server name to the DNS entry of the IP Alias

9.5.2 Connecting to a remote security service, such as LDAP

In case you are connecting to a remote LDAP directory server that resides on a cluster, you should also consider your maximum downtime. The default timeout value for WebSphere V6 waiting for a LDAP response is 120 seconds. In case your maximum downtime is greater than that *and* your application can handle a greater waittime, you can adjust this timeout value.

The other important setting in such a configuration scenario is again the Host name. Both settings are configured on the same panel in the Administrative Console, which can be reached via **Security** → **Global security** → **LDAP**. This panel is shown in Figure 9-18 on page 312.

Search timeout

The default value is to wait 120 seconds before WebSphere Application Server aborts the request.

Host name

Make sure that you provide the IP Alias or the host name assigned to that IP Alias. This is important as in case of a cluster switch, the LDAP node changes. Giving the IP Alias ensures that you can still access the LDAP service after a cluster switch. In our example the DNS name of the IP Alias is VCluster.ibmredbook.com.

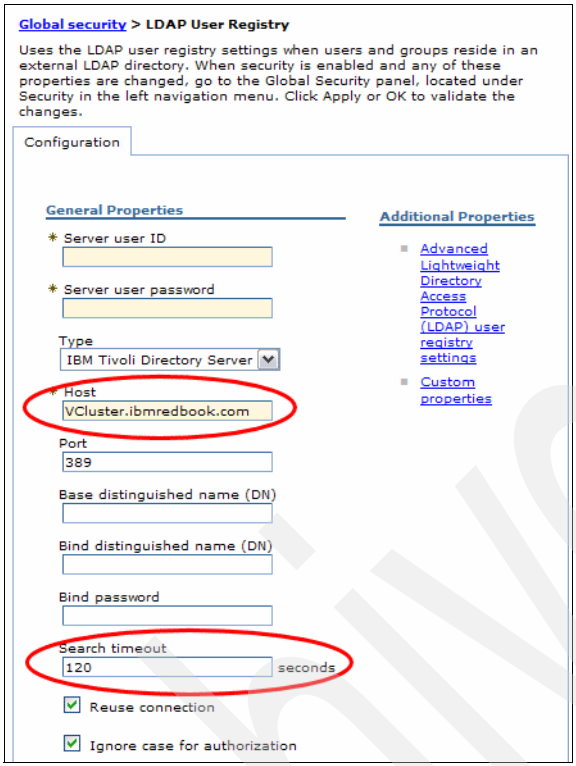


Figure 9-18 Configure the IP Alias and search timeout value for LDAP

9.5.3 Connecting to a remote messaging engine

If your messaging engine resides on a cluster, you must again consider your maximum downtime. The default timeout value for WebSphere V6 waiting for a new connection to a messaging engine is 180 seconds. In case your maximum downtime is greater than that *and* your application can handle a greater waittime, you can adjust this timeout value. Figure 9-19 on page 313 shows an example.

For further information about settings regarding high availability of messaging engines, see Part 5, “Messaging,” of the *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392.

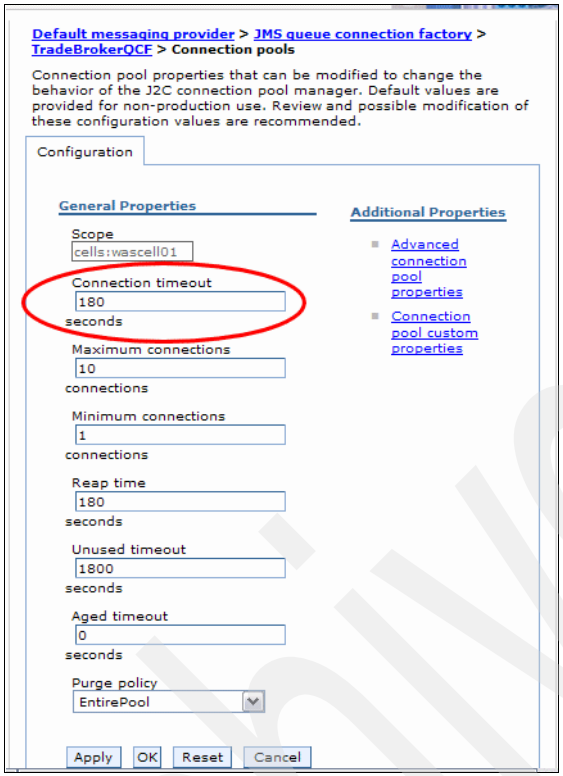


Figure 9-19 Configure the messaging engine connection timeout value

9.6 Transaction Manager failover with No Operation policy

WebSphere Application Server V6 has a new built in component called High Availability Manager (HAManager). This component is responsible for running key services on available servers. Prior to V6, these key services ran on dedicated servers and became single points of failure. Transaction Manager (TM) recovery is one of the key services that take advantage of the HAManager component to ensure that recovery of failed or in-doubt transactions can be done by other servers in the cluster.

The Transaction Manager supports three different HA policies to achieve the recovery of transaction logs in a highly available manner:

- One of N policy

  This is the default style of peer recovery initiation. If an application server fails, the HAManager selects another server to perform peer recovery processing on behalf of the failed server.

- Static policy

  This style of peer recovery must be explicitly configured. If an application server fails, the operator can use the Administrative Console to select another server to perform the recovery processing.

- No Operation policy

  This style of peer recovery must be explicitly configured. It indicates that external clustering software is monitoring the Transaction Manager and will failover to an application server that is configured by the external clustering software to perform the recovery processing.

Chapter 6, “WebSphere HAManager” on page 175, contains information about Transaction Manager failover and specifically the effect of the One of N and Static policies. In addition, the white paper *Transactional high availability and deployment considerations in WebSphere Application Server V6* provides an excellent description of the One of N and Static policies. You can find this paper at:

`http://www.ibm.com/developerworks/websphere/techjournal/0504_beaven/0504_beaven.html`

This section concentrates on how to use the No Operation (NoOP) policy for TM recovery.

The No Operation policy is used in situations where the customer requirements dictate great control over when and where transaction log recovery occurs. Some customers might want to store the WebSphere transaction logs on file systems that allow only one system to mount at a time, for example the Journal File System (JFS) of AIX. In such an environment, external clustering software is necessary to make the WebSphere solution highly available. The external HA software performs failure detection and controls the allocation of shared resources such as shared file systems and IP addresses.

9.6.1 Prerequisites for Transaction Manager with NoOP policy

There are several pre-requisites that need to be met prior to configuring the Transaction Manager with the NoOP policy:

- Shared file system

  In order for other cluster members to be able to perform transaction recovery, the cluster members must have access to the transaction logs of the failed server. If the cluster members are on different physical systems, then the file system must be accessible from each system.

  Another important consideration regarding the file system are the file locking semantics. When a cluster member becomes unavailable, then the peer cluster member that is elected to perform recovery must be able to exclusively lock the transaction log to ensure data integrity. See 6.7.2, “Hot-failover of Transaction Manager using shared file system” on page 206 for more information about shared file systems and the requirements for TM.

- External clustering software

  This software provides continuity to business applications. The software typically provides functionality such as IP takeover, monitoring of applications, starting and stopping applications, heartbeating and various other availability functions. There are many clustering software vendors with products such as IBM HACMP, Tivoli System Automation (TSA), Sun Cluster and VERITAS Cluster Server.

9.6.2 Transaction Manager with No Operation policy scenario

In order to explain how to configure the Transaction Manager peer recovery with the No Operation (NoOP) policy, we use the Trade 6 application in the topology shown in Figure 9-20. In this sample topology, the WebSphere hardware environment consists of two nodes and a shared file system. The cell contains one cluster with two cluster members, one on each node. Each cluster member has its own transaction log.

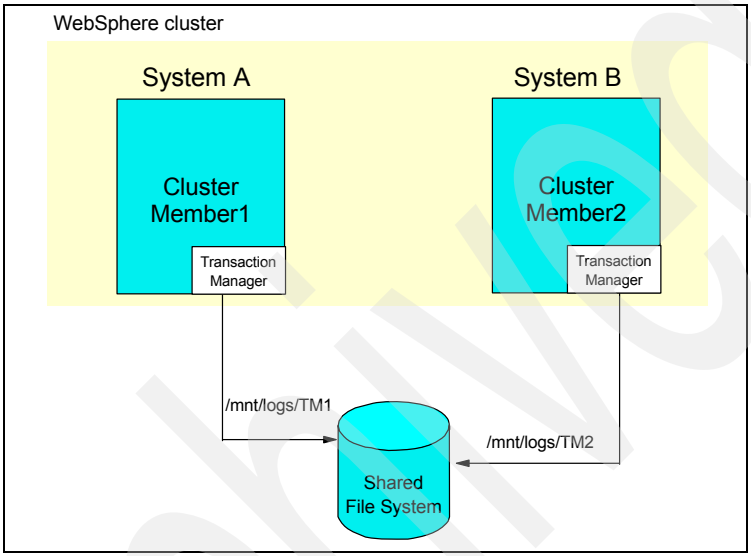


Figure 9-20 Sample topology for TM with NoOP policy

The scenario is as follows: one cluster member fails and the other cluster member takes over the transaction log to perform a peer recovery of in-doubt transactions. The NoOP policy indicates that external clustering software controls the actions of selecting *which* cluster member performs the transaction recovery and *when* this cluster member performs the recovery.

The example that we tested and that we describe here is an Active/Active solution for hot failover/log recovery. This means that both cluster members are actively running and processing transactions. When one cluster member fails, then the external clustering software directs the other cluster member to perform the recovery of the failing cluster member transaction log.

Our cluster name is TradeCluster, the cluster members are called TradeServer1 and TradeServer2.

The procedures needed to configure WebSphere and the external clustering software are described in the sections that follow.

9.6.3 Configuring WebSphere for TM No Operation policy

First, install WebSphere on your nodes. See 9.4.2, “Installing WebSphere Application Server Network Deployment” on page 306 for information. Then, create your cluster. For detailed instructions on how to set up clusters in WebSphere Application Server V6, see *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

When you have created your cluster, you need to enable the high availability option for the transaction logs and make the transaction logs available to all the cluster members. The recommended procedure to do this is as follows:

1. Stop the cluster members.
2. Enable high availability for the Transaction Manager.
3. Copy existing transaction logs to the new, shared directory.
4. Configure your servers to point to the new location for the transaction logs.
5. Create the No Operation policy for the Transaction Manager.

Enabling high availability for the Transaction Manager

After stopping the cluster members, you must tell the Transaction Manager that you want all cluster members to have the ability to recover each others transaction logs. To do this, go to **Servers** → **Cluster** → ***your_cluster*** and select **Enable high availability for persistent services (**Figure 9-21). Do not forget to save your changes.

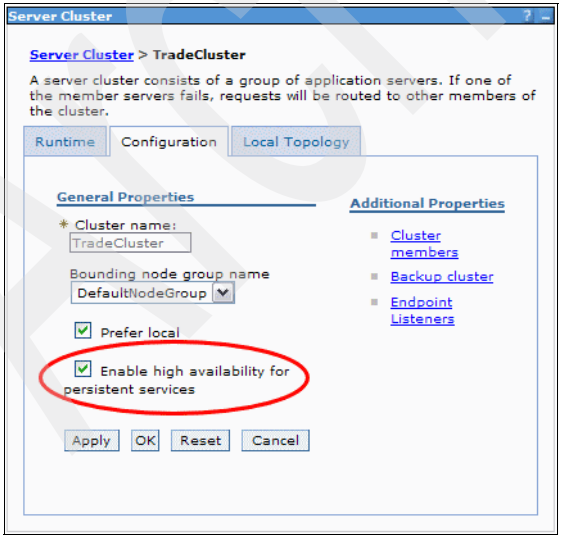


Figure 9-21 Example of enabling for transaction log high availability

The affect of selecting this option is that each cluster member joins the HA group associated with each cluster member's Transaction Manager. In our example, this means that TradeServer1 and TradeServer2 are both members of the HA group that is associated with the Transaction Manager for TradeServer1. Likewise, TradeServer1 and TradeServer2 are also members of the HA group associated with the Transaction Manager for TradeServer2. The net affect of this is that it enables each cluster member to be notified by the HAManager that it can perform peer recovery for another cluster member. In the NoOP policy, this notification is triggered by the external clustering software.

Copying existing transaction logs to the shared directory

After you have enabled the high availability for persistent services, you determine the location of the transaction logs so that they are accessible by each cluster member. There are several options available for how to perform this task, depending on the operation system and storage device that you are using.

In our example, we used a Network Attached Storage (NAS) device and a Linux operating system. We also used iSCSI to attach to the NAS and mounted the file system. Our environment allowed us to have simultaneous access to the file system from both systems in our topology.

In some environments, however, only one system can have mount access to the device at one time. In this case, you need to create a mount resource in your external clustering software and explicitly mount the shared device during the failover. We discuss this further in 9.6.4, "Configuring external clustering software for Transaction Manager No Operation policy recovery" on page 325.

For our example, we mounted our shared device onto the /shared file system and put the logs into a directory called logs. We created two transaction log directories, one for each cluster member: /shared/logs/TradeServer1 for the first cluster member's transaction log and /shared/logs/TradeServer2 for the second cluster member's transaction log.

When you have determined the location of the transaction logs, you have to move any existing logs into the appropriate directory for the cluster member on your shared file system. You also need to inform WebSphere about the location of the transaction logs on the shared file system.

To move the existing logs, go to the following directory:

<*install_root*>/WebSphere/AppServer/profiles/*profilename*/tranlog/*cellname*/*nodename*/*servername*/transaction

Copy both the tranlog and partnerlog subdirectories to the shared file system directory that you created for your transaction logs.

In our example, we copied these logs to the /shared/logs/TradeServer1/tranlog and /shared/logs/TradeServer1/partnerlog directories.

Pointing the TM to the new location for the transaction logs

After you have moved the existing transaction logs to the shared file system, you need to configure the location of each cluster members log in WebSphere. In the Administrative Console:

1. Go to **Servers** → **Application servers** → ***your_server*** → **Container Services** → **Transaction Service** as shown in Figure 9-22.

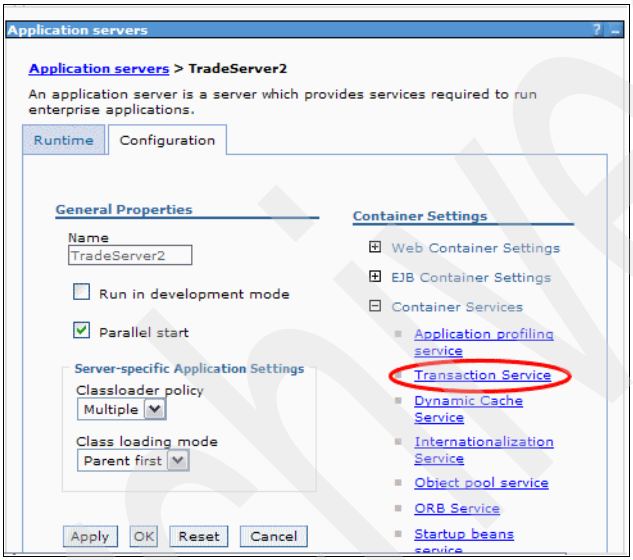


Figure 9-22 Way to configure the Transaction service

2. Enter the location of the transaction log on the shared file system into the Transaction Log Directory field. In our example for our first cluster member (TradeServer1), we entered the directory /shared/logs/TradeServer1 (see Figure 9-23). You have to set the log directory for each member of the cluster.

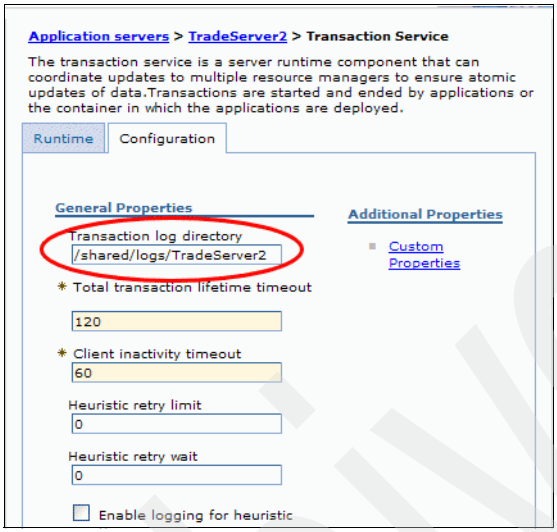


Figure 9-23 Specify the transaction log location

3. Save your changes and make sure the configuration is synchronized to all nodes.

Creating the No Operation policy for the Transaction Manager

The last configuration step is to create the No Operation policy for the Transaction Manager.

Policies are associated with core groups. You can either configure your own core groups or use the default one that is available for each cell. See 6.2, “Core group” on page 177 for a discussion on core groups. In our example, we assume that there is only the default core group.

HA groups and match criteria

The Transaction Manager registers himself as an HA group under a specific naming convention. In order to set the policy to correlate to the Transaction Manager, you must set policy match criteria that corresponds to the Transaction Manager HA group. Table 9-1 on page 321 lists the naming convention for the Transaction Manager HA group.

Table 9-1 TM HA group naming conventions

Name	Value
GN_PS	cellname\nodename\servername
IBM_hc	clustername
type	WAS_TRANSACTIONS

In our example, there are two HA groups associated with our Transaction Managers. The first HA group is associated with the Transaction Manager for our first cluster member (TradeServer1) and has the following HA group name:

```
GN_PS=myCell\myNode01\TradeServer1,IBM_hc=myCluster,type=WAS_TRANSACTIONS
```

The second HA group is associated with the Transaction Manager for our second cluster member (TradeServer2) and has the following HA group name:

```
GN_PS=myCell\myNode02\TraderServer2,IBM_hc=myCluster,type=WAS_TRANSACTIONS
```

The Administrative Console allows to see all HA groups that exist for your WebSphere environment. While the cluster members are started, go to the Administrative Console and click **Servers** → **Core groups** → **Core group settings** → **DefaultCoreGroup.** Select the Runtime tab, and click **Show groups**. This displays all existing HA groups and the name of each group. Remember that HA groups are only alive at runtime, so the cluster members must be started for this panel to show the desired content. Figure 9-24 on page 322 shows the HA group list for our environment.

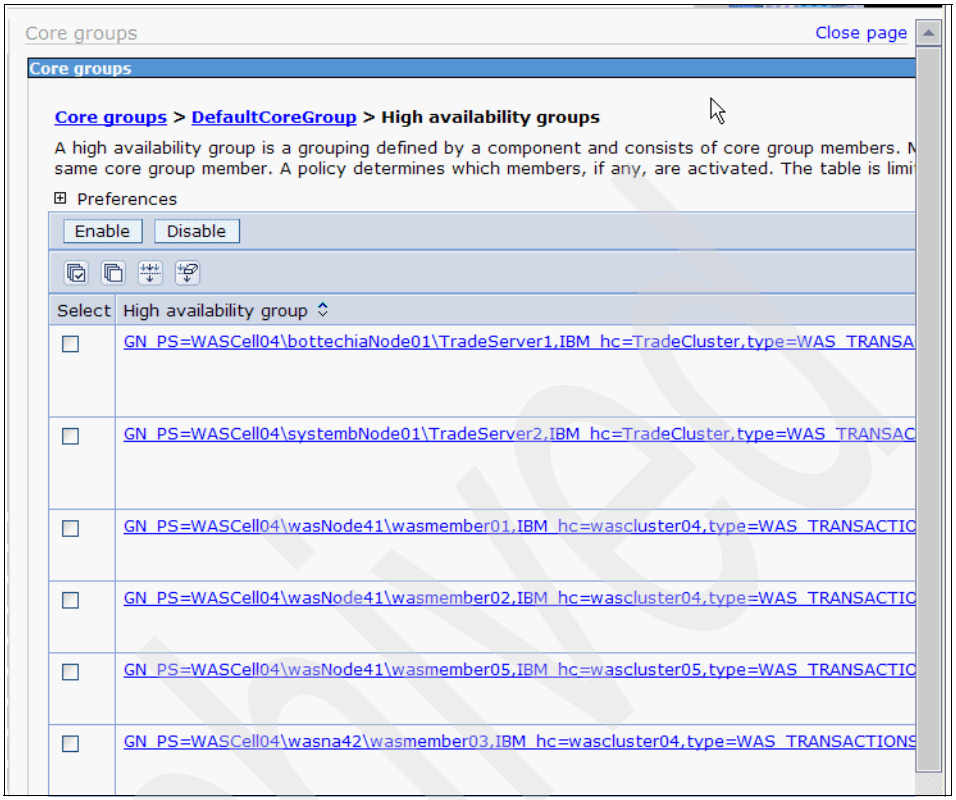


Figure 9-24 List of active HA groups

Creating the HA policy

Now that we understand how the Transaction Manager HA group is named, we need to define an HA policy that matches the HA group. To set the HA policy:

1. Go to **Servers** → **Core groups** → **Core group settings** → **DefaultCoreGroup** and click **Policies**.

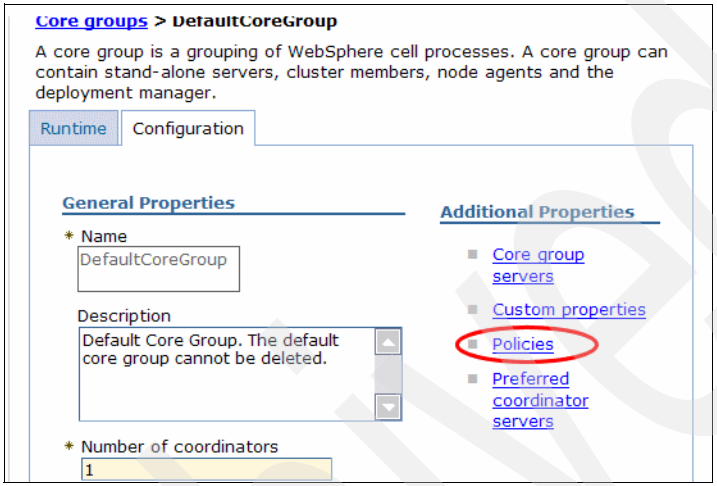


Figure 9-25 Create TM NoOP policy

2. Click **New** to create a new policy.
3. Select **No operation policy** as the policy type and click **Next**.
4. Enter a name for the policy. In our example, we named our policy Cluster TM Ext Cluster Sfw.
5. Click **OK**. Figure 9-26 on page 324 shows the resulting panel.

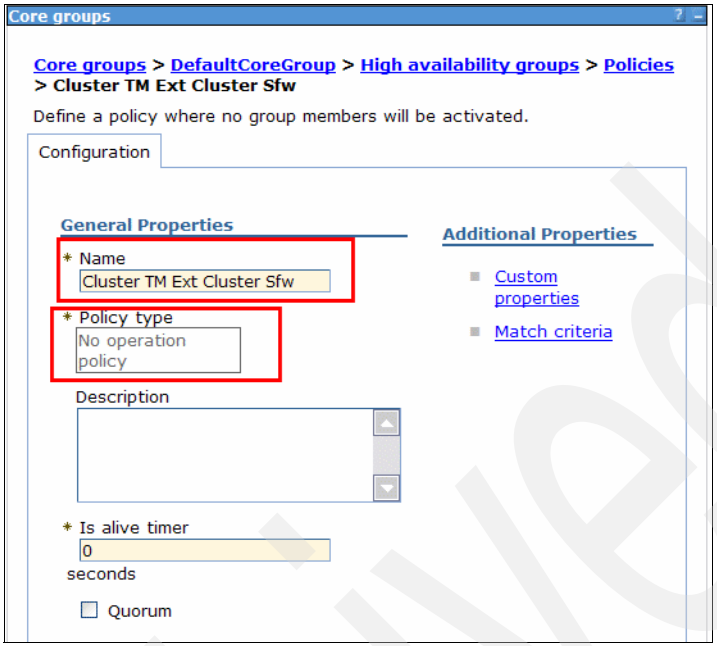


Figure 9-26 Create NoOP policy for TM HA group

Next click **Match criteria** in the Additional Properties section. This allows you to set the match criteria needed to associate this policy with an HA group. Specifically, we want to match the HA group for the Transaction Manager. If you refer back to the paragraph above on the naming convention for the Transaction Manager HA group, you can see that there are three parts to the name: GN_PS, IBM_hc, and type.

We want to define *one* policy that matches *all* Transaction Manager HA groups in the cluster. Therefore, we do not include the GN_PS part of the Transaction Manager HA group, because it has a value that is specific for each cluster member. If we were to include GN_PS as part of the match criteria, then we would have to define a NoOP policy for *each* cluster member. If we however do not include GN_PS then we can define one policy and have it match all the Transaction Manager HA groups in the cluster. Therefore, we include only the type and the IBM_hc values as match criteria for the new No Operation policy. In our example, the match criteria we want to define are `type=WAS_TRANSACTIONS` and `IBM_hc=TradeCluster`.

To do this:

1. Click **New**.
2. Enter `type` as the Name and `WAS_TRANSACTIONS` as the Value. Click **OK**.

3. Create the second match criteria by clicking **New** again.
4. Enter `IBM_hc` as the Name and `TradeCluster` as the Value. Click **OK**.

Figure 9-27 shows the result.

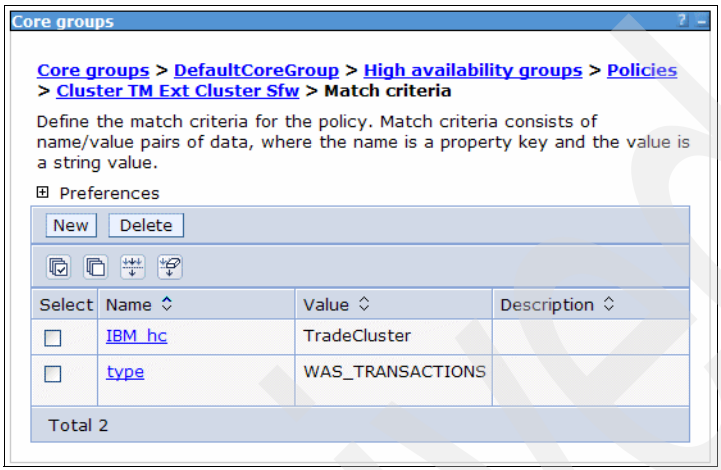


Figure 9-27 Create match criteria for TM NoOP policy

Save your changes and make sure your configuration is synchronized to all nodes.

We have finished configuring WebSphere for the Transaction Manager No Operation recovery policy. Now, we must configure the external clustering software to control the operations of the failover.

Important: Do not attempt to start your cluster members with the Transaction Manager No Operation policy defined until you have completed configuring the external clustering software!

9.6.4 Configuring external clustering software for Transaction Manager No Operation policy recovery

This section discusses how to configure external clustering software for the Transaction Manager No Operation policy recovery. The section does not describe any specific cluster software but discusses configuring a generic cluster software. The remaining chapters in this part of the book discuss how to configure the specific clustering software.

Terminology

In order to generically discuss how to configure any clustering software for the TM NoOP policy, we first describe a common set of clustering software components:

- Resource

  This is an entity that can failover. A resource has a type and is started, stopped and monitored. Typically, the resource types include an application type, an IP type, and a file system type. Some clustering software allows the definition of custom resource types.

- Failover unit

  This is a group of resources that make up an application and typically failover as a group. Some resources in the group are configured as critical and if one of these resources fails, the entire group fails over.

- Dependency

  This is a linkage between resources within a group and within groups. For example, a dependency can indicate the starting order of resources (that is: the file system must be mounted prior to the application starting) or indicate the starting order of groups (that is: application group A has to start before starting group B).

Clustering software tasks

When the TM is configured with the NoOP recovery policy, several functions must be coordinated with the external clustering software. These functions are:

- The startup of a cluster member that is configured with the TM NoOP recovery policy halts prior to completion of the start. The startup of the cluster member is halted until the HAManager is notified, via an MBean call, that it can continue. The reason for halting the cluster member startup allows the external clustering software to ensure all the dependant resources are online and available on the system that is hosting the cluster member. This allows any dependent file system to be mounted.
- Monitoring a specific cluster member's TM and when it becomes unavailable, moving any dependent resources to another system and then notifying the cluster member on the other system that it can perform the recovery for the cluster member that is unavailable.

These tasks are illustrated in Figure 9-28 on page 327.

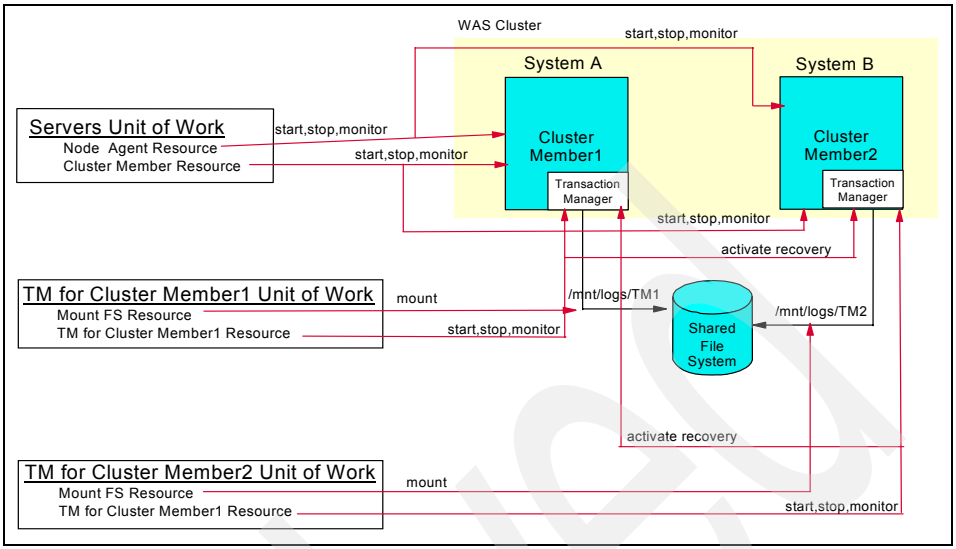


Figure 9-28 Failover units and resources actions

In order to achieve these functions, the external clustering software must be configured with the following units of work, resources and dependencies:

1. Start Server failover unit

   This failover unit is responsible for ensuring that the application server starts up and stays up on each system in the cluster. This failover unit is defined as an Active/Active failover unit which means that the failover units start the servers up simultaneously on each system in the cluster and that there is no failover from one system to another when a server is down. If the monitoring programs notice that a critical resource in not available, an attempt is made to restart it on the system that is defined to host the resource. This failover unit consists of two resources:

   a. Start, stop, and monitor the Node Agent. This resource is defined as an application resource. The start and stop programs are invocations of the WebSphere **`startNode`** and **`stopNode`** commands. The monitoring is done by either specifying the process ID that needs to be active or by monitoring a port that the Node Agent listens on to ensure that it is listening. The specific monitoring technique used depends on the functionality provided by the external clustering software and is thus described in the individual chapter for the cluster software. This resource starts, stops, and monitors the Node Agents on each system in the cluster. Typically, this resource is not deemed as critical to this failover unit, which means that the Node Agent can be unavailable and the clustering software does not attempt to restart it. This is because the cluster

members can still process runtime requests without the Node Agent being active.

b. Start, stop, and monitor the cluster member. This resource is defined as an application resource. As mentioned above, when a cluster member that has the TM NoOP recovery policy defined starts up, it goes into a halt state until the HAManager is notified. The objective of this resource is to initiate the starting of the cluster members and then return with a successful return code so the cluster software does not try to restart or failover the application server. In order to do this, the start, stop, and monitor programs are invocations of the wasctrl-as script (see “Scripts to start, stop, and monitor WebSphere resources” on page 331). This script invokes the **`startServer`** command with a timeout parameter so that it only waits for a specific amount of time and then ensures that a return code of success is returned. This resource has a dependency on the Node Agent resource which must have completed its starting before this resource can start.

2. TM for Cluster Member1 failover unit

   This failover unit cannot start until the Start Server failover unit has completed its start. This failover unit is defined as a Active/Passive or Primary/Backup resource. This means that it is only active on a single system at one time. When the monitoring determines that a resource is unavailable, then the failover unit is failed over to the Standby/Backup system. The primary system for this failover unit is the system where Cluster Member1 is hosted. This failover unit consists of two resources:

   a. Mount the dependant file system. This resource is defined as a file system resource. It mounts the file system where the Transaction Manager log is stored.

   b. Activate, deactivate, and monitor the TM for Cluster Member1. This resource is defined as an application resource. This resource uses the wasctrl-tm script to perform its functions. It has different invocation strings registered for the primary and the secondary systems. The primary system is the system hosting the Transaction Manager for the cluster member (in this case the Transaction Manager for Cluster Member1). The secondary system is the system hosting the cluster member that is defined to perform the peer recovery. This resource performs the following functions:

      i. It activates the Transaction Manager for the cluster member. If the cluster member is in initial start, the activate allows the cluster member to complete its startup processing. This action happens when the resource runs on the cluster members’ hosting system.

      ii. It activates the Transaction Manager recovery for another cluster member. If the primary cluster member fails, then this resource tells

another cluster member's Transaction Manager to perform the peer recovery of the transactions of the failed cluster member.

iii. It deactivates the Transaction Manager. If it runs on the primary system, then it does nothing. This is because the Transaction Manager does not allow itself to be deactivated when it is active on its primary system. If it is run against the secondary system, then the deactivate indicates that the Transaction Manager of the secondary cluster member is not to perform any more recovery.

iv. It monitors the status of the Transaction Manager.

3. TM for Cluster Member2 failover unit

   This failover unit cannot start until the Start Server failover unit has completed its start. This failover unit is defined as a Active/Passive or Primary/Backup resource. This means that it is only active on a single system at one time. When the monitoring determines that a resource is unavailable, then the failover unit is failed over to the Standby/Backup system. The primary system for this failover unit is the system where Cluster Member2 is hosted. This failover unit consists of two resources:

   a. Mount the dependant file system. This resource is defined as a file system resource. It mounts the file system where the Transaction Manager log is stored.

   b. Activate, deactivate and monitor the TM for Cluster Member2. This resource is defined as an application resource. This resource uses the wasctrl-tm script to perform its functions. It has different invocation strings registered for the primary and the secondary systems. The primary system is the system hosting the Transaction Manager for the cluster member (in this case the Transaction Manager for Cluster Member2). The secondary system is the system hosting the cluster member that is defined to perform the peer recovery. This resource performs the following functions:

      i. It activates the Transaction Manager for the cluster member. If the cluster member is in initial start, the activate allows the cluster member to complete its startup processing. This action happens when the resource runs on the cluster members' hosting system.

      ii. It activates the Transaction Manager recovery for another cluster member. If the primary cluster member fails, then this resource tells another cluster member's Transaction Manager to perform the peer recovery of the transactions of the failed cluster member.

      iii. It deactivates the Transaction Manager. If it runs on the primary system, then it does nothing. This is because the Transaction Manager does not allow itself to be deactivated when it is active on its primary system. If it is run against the secondary system, then the deactivate

indicates that the Transaction Manager of the secondary cluster member is not to perform any more recovery.

iv. It monitors the status of the Transaction Manager.

The chapters that follow provide examples of Transaction Manager No Operation policy configurations for a specific set of clustering software products (IBM HACMP, TSA, VERITAS Cluster Server, and Sun Cluster).

Monitoring resources

Most external clustering software requires the ability to monitor resources. In the context of these scenarios the desired resources to manipulate are the WebSphere Application Server highly available singleton services managed by the HAManager. Examples of such WebSphere groups or singletons are the Transaction Manager and messaging engines.

The singleton service resources can be manipulated in a variety of ways. The HAManager makes the singleton services/groups accessible via its MBeans. Among other functions the MBeans provide the ability to activate, deactivate, and query. See the HAManager.html document in <*install_root*>/web/mbeanDocs for more information.

Another approach is to use a **wsadmin** script to check if the group is active on the local cluster member but there are several problems with this approach. Ideally, this monitoring should have no impact on the running server, that is, it should use as little server resources as possible. The **wsadmin** utility starts a JVM and then loads many classes before actually executing the script. It can take as long as 20 to 30 seconds before **wsadmin** executes the script, and during this time, the utility might take 100% CPU. Obviously, this is not a lightweight monitor capability. It also takes around 30 seconds on an unloaded server. If one chooses to monitor every 10 seconds (or less) then this approach does not work.

Some of the overhead with **wsadmin** can be avoided by writing a Java JMX client to execute the same commands but this is still a JVM start and again many Java classes are loaded. It would still take several seconds of CPU time to run.

HAMonitorWeb application

Another alternative, very lightweight, approach for monitoring whether a group is active on the local cluster member is to use a servlet. We wrote a servlet which we deployed to the WebSphere cluster being monitored.

The HAMonitorWeb application (hamonitor.ear file) - together with the scripts explained in the next section - can be downloaded from the redbook repository. See Appendix B, “Additional material” on page 603 for download and installation instructions.

The servlet performs the following functions:

```
ActivateLocalGroupMember(String groupName, int state)
```

This sets the state of the local group member to either active (1) or idle (0).

```
QueryLocalGroupState(String groupName)
```

This returns the state for the local group for the specified group.

The servlet takes the name of the group to be used. The group name is a comma-delimited string with the group name. For example, the group name of a Transaction Manager group would look similar to the following:

```
GN_PS=myCell\myNode01\TraderServer1,IBM_hc=myCluster,type=WAS_TRANSACTIONS
```

This example is the group name for the Transaction Manager resource for cluster member TraderServer1 in the cluster myCluster. For more information about HA group names see Table 9-1 on page 321.

As part of the lightweight monitoring, the servlet can be invoked by a Perl script. In our case, the servlet is invoked by the support scripts mentioned below.

The benefit of Perl is almost zero startup time and processor usage during the monitor operation. This monitor can be used with high frequency while putting very low load on the server.

Scripts to start, stop, and monitor WebSphere resources

To interact with the HAManager singleton services and other associated WebSphere Application Server resources, we wrote a set of generic scripts that most external clustering software can use without much variation.

In total there are five scripts, which are listed in Table 9-2 on page 332. They can be separated into two categories:

- Main control scripts
- Support scripts

The control scripts perform all of the main control and call the underlying support scripts for minor activities. Although we give details of their function, the clustering software does not directly invoke the support scripts.

Table 9-2 Scripts listing

	Name	Description
Example 9-4	wasctrl-tm	Starts, stops, or returns status on a HAManager resource group.
Example 9-5	wasctrl-as	Starts, stops, or returns status on an application server instance.
Example 9-6	controlHA.pty	The **wsadmin** script invoked by wasctrl-tm. Although accessible, this script is not normally called by the user.
Example 9-7	activategroup.pl	Attaches to the HTTP client on the local cluster member of the server and calls the ActivateLocalGroupMember method.
Example 9-8	deactivategroup.pl	Attaches to the HTTP client on the local cluster member of the server and calls the ActivateLocalGroupMember() method.
Example 9-9	monitorgroup.pl	Attaches to the HTTP client on the local cluster member of the server and calls QueryLocalGroupState() method.

Main control scripts

The main script for HA control is known as wasctrl-tm. The script takes a set of parameters that can be broken down into a hierarchy of purposes. The categories are Action, Connectivity, Match Criteria, and Host information. The action category indicates the action the script should do when it runs. Connectivity relates to the TCP network connectivity. Match criteria are the name/value pairs for IBM_hc and type. The fully qualified match criteria consist of the name/value pairs along with the cell name, node name, server name described in the Host info parameters. See Table 9-3 on page 333 for a description of wasctrl-tm.

Table 9-3 wasctrl-tm

Name	wasctrl-tm This script starts, stops, or returns the status of a HAManager resource group.
Example	`wasctrl-tm start /opt/IBM/WebSphere/AppServer/profiles/wasNode01/ 9080 systemb 8879 IBM_hc=TradeCluster,type=WAS_TRANSACTIONS WASCell04 TradeServer2 /usr/bin/WAS_HA_SCRIPTS/ systembNode01`
Parameter Description	► Action: The first parameter indicates whether to start, stop, or monitor the HA resource group. ► Connectivity: The third and fourth parameters are the host name and port for the servlet that is installed on the cluster member to perform the MBean actions. The fifth parameter is the **wsadmin** binding (SOAP) port. When activating against a cluster member that is not started (the primary cluster member), the script uses a JACL script and this is the SOAP port against which to run **wsadmin** and the JACL script. ► Match Criteria: The sixth parameter is part of the match criteria for selecting which TM to work against. ► Host info: The second parameter is the install directory for the node. The seventh parameter is the WebSphere cell name. The eight parameter is the name of the application server. The tenth parameter is the name of the node that hosts the WebSphere Application Server. The seventh, eight and tenth parameter are actually part of the match criteria and must always point to the TM's original cell/node/application server. The ninth parameter is a pointer to where the scripts are installed.
Return values	The script prints different values depending on the action specified: ► start – 0 returned if started ok – 42 returned if there was a problem ► stop – 0 returned if started ok – 42 returned if there was a problem ► monitor – 110 returned if the resource is online – 100 returned if the resource is offline

The main script for WAS application server control is known as wasctrl-as and is explained in Table 9-4.

Table 9-4 wasctrl-as

Name	wasctrl-as This script starts, stops, or returns the status of an application server instance.
Example	`start /opt/IBM/WebSphere/AppServer/profiles/wasNode01/ 9080 TradeServer2`
Parameter Description	The first parameter is the action and indicates to start the application server. The second parameter is the installation directory of the WebSphere node. The third parameter is the WebSphere Application Server's listening port. It is queried to determine if the application server is available. The fourth parameter is the name of the WebSphere application server to act upon. Be advised that case matters when specifying the application server name.
Return values	The script prints different values depending on the action specified. ► start – 0 returned if started ok ► stop – 0 returned if stopped ok – 246 returned if there was a timeout problem and the stop server never executed ► monitor – 110 returned if the resource is online – 100 returned if the resource is offline

Support scripts

There are four support scripts, controlHA.pty, activategroup.pl, deactivategroup.pl, and monitorgroup.pl. These scripts are explained in the tables that follow.

Table 9-5 controlHA.pty

Name	controlHA.pty
Example	`wsadmin.sh -conntype SOAP -host TradeServer2 -port $HOST_PORT -lang jython -f /usr/bin/WAS_HA_SCRIPTS/controlHA.pty activate ms=GN_PS=WASCell04\systembNode01\systemb,IBM_hc=TradeCluster,type=WAS_TRANSACTIONS`
Parameter Description	There are two sets of parameters involved in invoking the controlHA script. The first set are for **wsadmin**, the second set are the parameters used by the script itself. ▶ **wsadmin** parameters The first parameter is the type of connection to use with **wsadmin**. The second parameter is the host. The third parameter is the port. The fourth parameter is the type of script being passed to **wsadmin**. The fifth parameter is the fully qualified name of the script. ▶ Script parameters The first parameter is the action to invoke on the HA group. The second parameter is criteria used to find the proper HA group.

Table 9-6 activategroup.pl

Name	activategroup.pl
Example	`perl activategroup.pl WASCell04 wasNode01 TradeCluster TradeServer2 9080 IBM_hc=myCluster,type=WAS_TRANSACTIONS`
Parameter Description	The first parameter is the WebSphere cell name. The second parameter is the WebSphere node name. The third parameter is the WebSphere cluster name. The fourth parameter is the WebSphere application server name. The fifth parameter is the WebSphere application server listening port. The sixth parameter is the match criteria for the group.

Table 9-7 deactivategroup.pl

Name	deactivategroup.pl
Example	`perl deactivategroup.pl WASCell04 wasNode01 TradeCluster TradeServer2 9080 IBM_hc=myCluster,type=WAS_TRANSACTIONS`
Parameter Description	The first parameter is the WebSphere cell name. The second parameter is the WebSphere node name. The third parameter is the WebSphere cluster name. The fourth parameter is the WebSphere application server name. The fifth parameter is the WebSphere application server listening port. The sixth parameter is the match criteria for the group.

Table 9-8 monitorgroup.pl

Name	monitorgroup.pl
Example	`perl monitorgroup.pl WASCell04 wasNode01 TradeCluster TradeServer2 9080 IBM_hc=myCluster,type=WAS_TRANSACTIONS`
Parameter Description	The first parameter is the WebSphere cell name. The second parameter is the WebSphere node name. The third parameter is the WebSphere cluster name. The fourth parameter is the WebSphere application server name. The fifth parameter is the WebSphere application server listening port. The sixth parameter is the match criteria for the group.

The scripts

This section shows the full scripts. You can download these scripts from the redbook repository. See Appendix B, “Additional material” on page 603 for instructions on how to obtain the scripts.

Example 9-4 shows the wasctrl-tm script.

Example 9-4 wasctrl-tm

```
#!/bin/ksh
############################################################
#
# was HA service automation control script
#
# Input:
# $1      action (<start|stop|status>)
# $2      na_home, the nodeagent's home directory USER_INSTALL_ROOT
# $3      as_tcpp, the applicationServer's listening port (WC_defaultHost)
# $4      host_name, name of the system that is hosting the application server
# $5      host_port, SOAP_CONNECTOR_ADDRESS port of the application server
# $6      matchingCriteria, HA policy matching criteria for the NOOP Policy
```

```
# $7      cell_name, name of the cell that is hosting the application server
# $8      servername, name of the application server or cluster member.
# $9      script_path, directory that contains the scripts (Perl scripts and
tmHA.pty)
# $10     nodename, name of the WAS node that is hosting the application server
(for ex: mySystemNode01)
#
############################################################
#
# init section
#

UNKNOWN=0
ONLINE=110
OFFLINE=100

Action=${1:-status}
NA_HOME=$2
AS_TCPP=$3
HOST_NAME=$4
HOST_PORT=$5
CRITERIA=$6
CELL_NAME=$7
SERVER_NAME=$8
SCRIPT_PATH=$9
NODENAME="${10}"


export
PATH=$PATH:/bin:/usr/bin:/sbin:/usr/sbin:/usr/sbin/rsct/bin:${NA_HOME}/bin

#
#



case ${Action} in
        start)
                AS_UP=`netstat -lnt | grep :${AS_TCPP}`
                 if [ "${AS_UP}" != "" ]; then
                    print  "Application Server is up so run the perl script"
                    RetC=`perl $SCRIPT_PATH/activategroup.pl ${CELL_NAME}
${NODENAME}  ${SERVER_NAME} ${AS_TCPP} ${CRITERIA}|awk '{print $1}'`
                     # rcs must be tested!!
                    if [ $RetC == "<OK/>" ]; then
                      RC=0
                    else
                      RC=42
                    fi
```

```
                   print "Transaction manager start rc: ${RC}"
                fi
                ;;
        stop)
                RetC=`perl $SCRIPT_PATH/deactivategroup.pl ${CELL_NAME}
${NODENAME}  ${SERVER_NAME} ${AS_TCPP} ${CRITERIA} |awk '{print $1}'`
                # rcs must be tested!!
                if [ $RetC == "<OK/>" ]; then
                  RC=0
                else
                  RC=42
                fi
                print "Transaction manager stop rc: ${RC}"
                ;;
        status)
                print "Requesting status"
                RetC=`perl $SCRIPT_PATH/monitorgroup.pl ${CELL_NAME}
${NODENAME}  ${SERVER_NAME} ${AS_TCPP} ${CRITERIA} `
                # rcs must be tested!!
                print "Status = ${RetC}"
                if [ $RetC == "<ACTIVE/>" ]; then
                  RC=$ONLINE
                else
                  RC=$OFFLINE
                fi

                print "Transaction manager status rc: ${RC}"
                ;;
        *)
                print "Error: Incorrect parameter >${Action}<"
                RC=${UNKNOWN}
                ;;
esac

exit ${RC}
```

Example 9-5 shows the wasctrl-as script.

Example 9-5 wasctrl-as

```
#!/bin/ksh
############################################################
#
# was applicationServer automation control script
#
# Input:
#       $1      action (<start|stop|status>)
#       $2      na_home, the nodeagent's home directory
#               USER_INSTALL_ROOT
#       $3      as_tcpp, the applicationServer's listening port
#       $4      server_name, name of the applicationServer
#
############################################################
#
# init section
#

SCEN3="yes"

UNKNOWN=0
ONLINE=110
OFFLINE=100

Action=${1:-status}
NA_HOME=$2
AS_TCPP=$3
SERVER_NAME=$4

export
PATH=$PATH:/bin:/usr/bin:/sbin:/usr/sbin:/usr/sbin/rsct/bin:${NA_HOME}/bin

#
#

case ${Action} in
       start)
               startServer.sh ${SERVER_NAME} -timeout 180 > /dev/null 2>&1
               RC=0;
               print "ApplicationServer start rc: ${RC}"
               ;;
       stop)
               stopServer.sh ${SERVER_NAME} -timeout 180 -trace > /dev/null
2>&1
               RC=$?
               if [ $RC -eq 0 ]; then
```

```
                  print "$ApplicationServer stop rc: ${RC}"
                elif [ $RC -eq 246 ]; then
                  print "$ApplicationServer stop rc: ${RC} (applicationServer
did not run)."
                else
                  print "$ApplicationServer stop rc: ${RC}"
                  AS_pid=`cut -f1
$NA_HOME/logs/${SERVER_NAME}/${SERVER_NAME}.pid`
                  kill -9 ${AS_pid} > /dev/null 2>&1
                  print "$ApplicationServer (pid ${AS_pid}) killed."
                fi
                ;;
        status)

                AS_UP=`netstat -lnt | grep :${AS_TCPP}`
                if [ "${AS_UP}" != "" ];
                then
                  RC=$ONLINE
                else
                  RC=$OFFLINE
                fi
                ;;
        *)
                print "Error: Incorrect parameter >${Action}<"
                RC=${UNKNOWN}
                ;;
esac
print "exiting with return code >${RC}<"
exit ${RC}
```

Example 9-6 shows the controlHA.pty support script.

Example 9-6 controlHA.pty

```
#This jython script is used to activate a HA service being managed by an HA
cluster. The HA cluster decides which machine the HA service should be
activated on and runs a script on that machine. The script invoked by the HA
cluster should be a shell script (bat file) that runs wsadmin targetting the
server that it wants the HA service to run on. This is achieved by setting the
host and port in the wsadmin conntype. The shell script (bat file) therefore
needs to be specialised on each node in the HA cluster.

# For example: Two machines in a cluster each have a shell script called
"controlscript"  and each wants to activate an HA singleton service
# # <-------------- "controlscript" ------------>
# machine 1: invoke wsadmin on server1's host and port : wsadmin -conntype SOAP
-host mach1 -port 8880 -lang jython controlHA.pty activate
IBM_hc=myCluster,type=WAS_TRANSACTIONS
```

```
# The controlHA will activate <meName> in the server to which wsadmin is
connected

# machine 2: invoke wsadmin on server2's host and port: wsadmin -conntype SOAP
-host mach2 -port 8881 -lang jython controlHA.pty activate
IBM_hc=myCluster,type=WAS_TRANSACTIONS
# The controlHA script will activate <meName> in the server to which wsadmin is
connected

#The controlHA script is passed the name of the cluster and the group type
"WAS_TRANSACTIONS" to indicate transaction manager and starts the service in
the server process to which wsadmin is connected. It does this by finding the
local HAManager MBean and calling the relevant operation on the MBean, e.g. for
action==start, it calls activateMember()

from com.ibm.ws.hamanager import AttributeNamesMemberProps
# Print usage information
def printHelp():
       print "Usage: controlHA <action> <matchingcriteria> where action is one
of {start|stop|monitor} and matchingcriteria is the HA policy matching criteria
to perform the action against (for example
IBM_hc=myCluster,type=WAS_TRANSACTIONS)"
       return

# Get the HAManager MBean for the specified server.
def getServerHAMProxy(servername):
       strObjectName =
AdminControl.queryNames("type=HAManager,process="+servername+",*")
       objectName = AdminControl.makeObjectName(strObjectName)
       mb = TypedProxy.makeProxy(AdminControl, objectName,
"com.ibm.websphere.hamanager.jmx.CoordinatorJMX")
       return mb
# Find the group for the specified service and the status of its members
def getHAGroup(matchingCrit):
       global gHAM
   matchSet="GN_PS="+cell+"\\"+node+"\\"+server+","+matchingCrit
       print matchSet
       gs={}
       gs = gHAM.queryGroupState(matchSet,1,1)

   #Should be only one HAGroup that matches.
   if (len(gs) < 1):
            print "There is no HAGroup for "+matchingCrit
            return {}
       elif (len(gs) > 1):
          print "There are multiple HAGroups for  "+matchingCrit

          return {}
   else:
```

```
            # There is one HAGroup for the match criteria
            groupName = gs[0].getGroupName()
return groupName

#
# START
#

global gHAM
if (len(sys.argv) ==2):
        action = sys.argv[0]
        matchingCrit = sys.argv[1]

else:
        printHelp()
        sys.exit(1)

# Find out which cell and node and server we are running in.
node = AdminControl.getNode()
cell = AdminControl.getCell()

print "node is "+node
print "cell is "+cell
serverMBean = AdminControl.queryNames('processType=ManagedProcess,*')
server = AdminControl.getAttribute(serverMBean,'name')

print "server is "+server

# Connect to the HAM MBean in the specified server
# If that server is not running then the operation won't work anyway so give up
gHAM = {}
gHAM = getServerHAMProxy(server)
if (gHAM == {}):
   print "Cannot connect to HAManager MBean, giving up"
   sys.exit(1)

# Locate the HAGroup for the service
haGroup = {}
haGroup = getHAGroup(matchingCrit)
if (haGroup == {}):
        print "Cannot identify HAGroup for  "+matchingCrit+": giving up"
   sys.exit(1)

if (action == 'activate'):
        print "Attempting to activate " +matchingCrit+" on server
"+node+"."+server
        # Activate the local member
        gHAM.activateMember(haGroup,node,server)
elif (action == 'deactivate'):
```

```
        print "Attempting to deactivate "+matchingCrit+" on server
"+node+"."+server
        # Activate the local member
        gHAM.deactivateMember(haGroup,node,server)
elif (action == 'monitor'):
        print "Monitor action not yet implemented!!!"
else:
        printHelp()
        sys.exit(1)
```

Example 9-7 shows the activategroup.pl support script.

Example 9-7 activategroup.pl

```
#
# Invocation:
# perl activategroup.pl cellname nodename clustername servername [portnumber]
#     where:
#              cellname is the name of the cell that contains the application
server or cluster member
#              nodename is the name of the node that contains the application
server or cluster member
#              servername is the name of the server or cluster member that
requires the group to be activated.
#              portnumber is the port number of the application that is doing
the activation.
#             matching is the HA matching criteria for the policy (for example
IBM_hc=myCluster,type=WAS_TRANSACTIONS)
#

require HTTP::Request;
require LWP::UserAgent;
use Sys::Hostname;

if (@ARGV != 5) {
   print "Illegal number of arguments";
} else {
  $CELLNAME= $ARGV[0];
  $NODENAME = $ARGV[1];
  $SERVERNAME = $ARGV[2];
  $PORTNAME = $ARGV[3];
  $MATCHING = $ARGV[4]
}

my $hostname = hostname();
my $getrequest =
'http://'.$hostname.":".$PORTNAME."/HAMonitorWeb/ActivateLocalGroupMember?ms=GN
_PS=".$CELLNAME."\\".$NODENAME."\\".$SERVERNAME.",".$MATCHING."&s=a";
```

```
$request = HTTP::Request->new(GET => $getrequest);

$ua = LWP::UserAgent->new();
$response = $ua->request($request);
if ($response->is_success) {
  $result = $response->content;
  if(index($result, "<OK/>", 0) >= 0)
  {
    print "<OK/>";
  }
  else
  {
    print "<NOT OK/>";
  }
}
else
{
  print "1";
  print STDERR $response->status_line, "\n";
}
```

Example 9-8 shows the deactivategroup.pl support script.

Example 9-8 deactivategroup.pl

```
#
# Invocation:
# perl deactivategroup.pl cellname nodename clustername servername [portnumber]
#     where:
# cellname is the name of the cell that contains the application server or
cluster member
# nodename is the name of the node that contains the application server or
cluster member
# servername is the name of the server or cluster member that requires the
group to be deactivated.
# portnumber is the port number of the application that is doing the
deactivation.
# matching is the HA matching criteria for the policy (for example
IBM_hc=myCluster,type=WAS_TRANSACTIONS)
#


require HTTP::Request;
require LWP::UserAgent;
use Sys::Hostname;

if (@ARGV != 5) {
   print "Illegal number of arguments";
} else {
```

```
  $CELLNAME= $ARGV[0];
  $NODENAME = $ARGV[1];
  $SERVERNAME = $ARGV[2];
  $PORTNUMBER = $ARGV[3];
  $MATCHING = $ARGV[4]
}

my $hostname = hostname();
my $getrequest =
'http://'.$hostname.":".$PORTNUMBER."/HAMonitorWeb/ActivateLocalGroupMember?ms=
GN_PS=".$CELLNAME."\\".$NODENAME."\\".$SERVERNAME.",".$MATCHING."&s=d";

$request = HTTP::Request->new(GET => $getrequest);

$ua = LWP::UserAgent->new();
$response = $ua->request($request);
if ($response->is_success) {
  $result = $response->content;
  if(index($result, "<OK/>", 0) >= 0)
  {
    print "<OK/>";
  }
  else
  {
    print "<NOT OK/>";
  }
print $result;
}
else
{
  print "1";
  print STDERR $response->status_line, "\n";
}
```

Example 9-9 shows the monitorgroup.pl support script.

Example 9-9 monitorgroup.pl

```
#
# Invocation:
# perl monitorgroup.pl cellname nodename clustername servername [portnumber]
#     where:
# cellname is the name of the cell that contains the application server or
cluster member
# nodename is the name of the node that contains the application server or
cluster member
# servername is the name of the server or cluster member that requires the
group to be activated.
```

```
# portnumber is the port number of the application that is doing the
activation.
# matching is the HA matching criteria for the policy (for example
IBM_hc=myCluster,type=WAS_TRANSACTIONS)
#


require HTTP::Request;
require LWP::UserAgent;
use Sys::Hostname;

if (@ARGV != 5) {
   print "Illegal number of arguments";
} else {
  $CELLNAME= $ARGV[0];
  $NODENAME = $ARGV[1];
  $SERVERNAME = $ARGV[2];
  $PORTNUMBER = $ARGV[3];
  $MATCHING = $ARGV[4]
}

my $hostname = hostname();
my $getrequest =
'http://'.$hostname.":".$PORTNUMBER."/HAMonitorWeb/QueryLocalMemberState?ms=GN_
PS=".$CELLNAME."\\".$NODENAME."\\".$SERVERNAME.",".$MATCHING."&s=a";

$request = HTTP::Request->new(GET => $getrequest);

$ua = LWP::UserAgent->new();
$response = $ua->request($request);
if ($response->is_success) {
  $result = $response->content;
  if(index($result, "<ACTIVE/>", 0) >= 0)
  {
    print "<ACTIVE/>";
  }
  else
  {
    print "<INACTIVE/>";
  }
}
else
{
  print "1";
  print STDERR $response->status_line, "\n";
}
```

9.7 Default messaging provider failover with No Operation policy

The messaging engine (ME) component of the default messaging provider is one of the key services that take advantage of the HAManager to ensure that failover of failed messaging engines can occur to other servers in the cluster. The default messaging provider supports three different HA policies to achieve the failover of a ME in a highly available manner:

- One of N policy

  This is the default style of peer recovery initiation. If an application server fails, the HAManager selects another server to perform peer recovery processing on behalf of the failed server.

- Static policy

  This style of peer recovery must be explicitly configured. If an application server fails, the operator can use the Administrative Console to select another server on which to activate the ME.

- No Operation policy

  This style of peer recovery must be configured explicitly. It indicates that external clustering software is monitoring the messaging engine and will failover to an application server that is configured by the external clustering software to perform the further messaging processing.

Chapter 12 of the *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392 and the WebSphere InfoCenter contain information about messaging engine failover and, specifically, the effect of the One of N and Static policies. This section concentrates on how to use the No Operation policy for ME failover.

The No Operation (NoOP) policy is used in situations where customer requirements dictate great control over when and where messaging engine failover occurs.

9.7.1 Prerequisites for default messaging provider with NoOP policy

You need to use external clustering software in addition to WebSphere Application Server in order to make this work.

This software provides continuity to business applications. The software typically provides functionality such as IP takeover, monitoring of applications, starting and stopping applications, heartbeating and various other availability functions.

There are many clustering software vendors with products such as IBM HACMP, Tivoli System Automation (TSA), Sun Cluster, and VERITAS Cluster Server.

9.7.2 Default messaging provider with No Operation policy scenario

In order to explain how to configure the default messaging provider with the No Operation (NoOP) policy, we use the Trade 6 application in the WebSphere topology shown in Figure 9-29. In this sample topology, the WebSphere environment consists of two nodes, the cell contains one cluster with two cluster members, one on each node. Our cluster name is TradeCluster, the cluster members are called TradeServer1 and TradeServer2. The Trade 6 application configuration script configures a default messaging provider messaging engine for each cluster member.

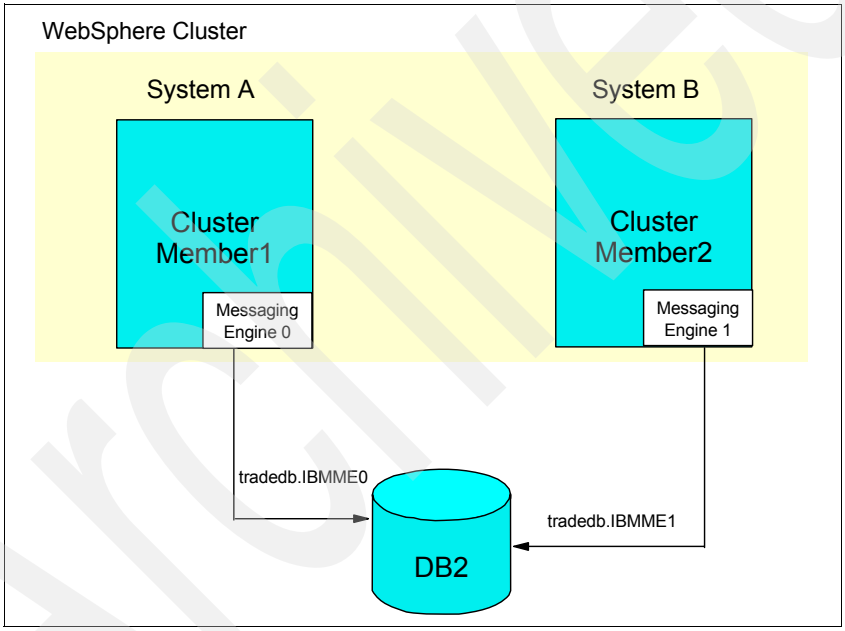


Figure 9-29 Sample topology for default messaging provider with NoOP policy

The example that we tested and that we describe here is an Active/Active solution for hot failover. This means that both cluster members are actively running and processing transactions. When the messaging engine in one cluster member fails, then the external clustering software directs the other cluster member to activate the failed messaging engine.

The procedures needed to configure WebSphere and the external clustering software are described in the sections that follow.

9.7.3 Configuring WebSphere for default messaging provider No Operation policy

First, install WebSphere on your nodes. See 9.4.2, “Installing WebSphere Application Server Network Deployment” on page 306 for information. Then, create your cluster. For detailed instructions on how to set up clusters in WebSphere Application Server V6, see *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

We used the script that comes with the Trade 6 application to create all necessary resources, including the default message provider messaging engines, and to install the application. This script, by default, sets up one ME per cluster member. For details on configuring default messaging provider messaging engines, see Chapter 11 of the *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451, or Chapter 12 of the *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392.

Creating the No Operation policy for the messaging engines

The configuration step that we discuss in this section is how to configure the No Operation policy for the default messaging provider messaging engines. As mentioned, during the Trade 6 configuration, two messaging engines are created and configured. They are called IBMME0 and IBMME1.

We create the NoOP policy in a way that one policy covers both messaging engines. Policies are associated with core groups. You can either configure your own core groups or use the default one that is available for each cell. See 6.2, “Core group” on page 177 for a discussion on core groups. In our example, we assume that there is only the default core group.

HA groups and match criteria

Each messaging engine of the default messaging provider registers himself as an HA group under a specific naming convention. In order to set the policy to correlate to the messaging engines, you must set policy match criteria that corresponds to the messaging engines HA groups. The naming convention for the messaging engines HA groups is shown in Table 9-9 on page 350.

Table 9-9 Messaging engines naming convention

Name	Value
WSAF_SIB_BUS	Name of the default messaging provider bus
WSAF_SIB_MESSAGING_ENGINE	Name of the messaging engine
IBM_hc	clustername
type	WSAF_SIB

For our example, there are two HA groups associated with our messaging engines. The first HA group (for TradeServer1) is associated with the messaging engine named TradeCluster.000-TradeCluster and has a HA group name of

```
IBM_hc=TradeCluster,WSAF_SIB_BUS=TradeCluster,WSAF_SIB_MESSAGING_ENGINE=
TradeCluster.000-TradeCluster,type=WSAF_SIB
```

The second HA group (for TradeServer2) is associated with the messaging engine named TradeCluster.001-TradeCluster and has a HA group name of

```
IBM_hc=TradeCluster,WSAF_SIB_BUS=TradeCluster,WSAF_SIB_MESSAGING_ENGINE=
TradeCluster.001-TradeCluster,type=WSAF_SIB
```

The Administrative Console allows you to see all HA groups that exist for your WebSphere environment. While the cluster members are started, go to the Administrative Console, and click **Servers** → **Core groups** → **Core group settings** → **DefaultCoreGroup.** Select the Runtime tab, and click **Show groups**. An example of this view is shown in Figure 9-30 on page 351. This displays all existing HA groups and the name of each group. Remember that HA groups are only alive at runtime, so the cluster members must be started for this panel to show the desired content. See 6.3, “High availability group” on page 194 for more information about HA groups.

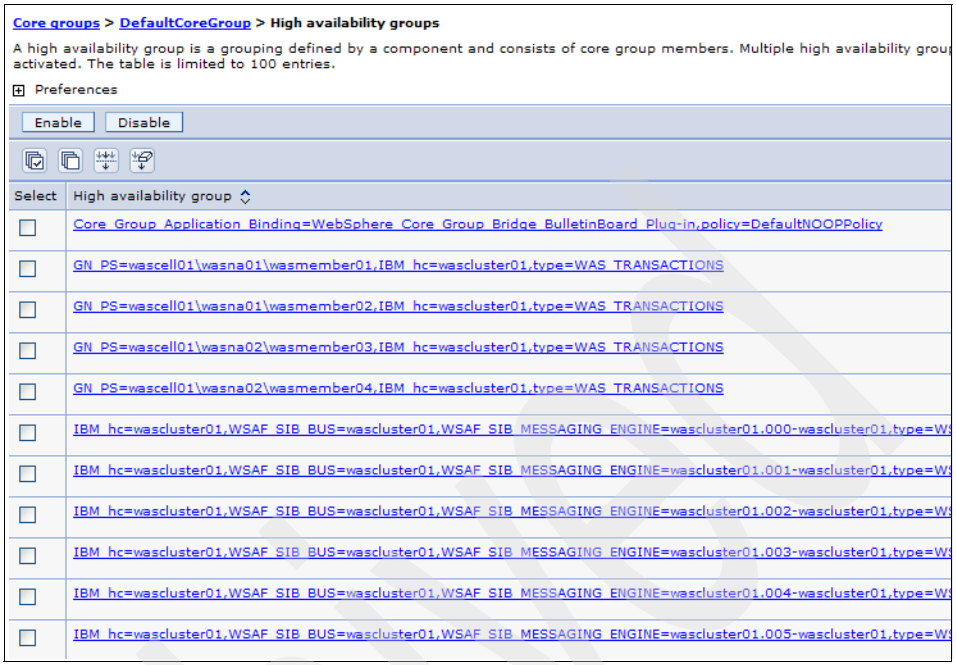


Figure 9-30 List of active HA groups

Creating the HA policy

Now that we understand how the messaging engine HA groups are named, we need to define an HA policy that matches the HA groups. To create the HA policy:

1. Go to **Servers** → **Core groups** → **Core group settings** → **DefaultCoreGroup** and click **Policy**.
2. Click **New** to create a new policy.
3. Select **No operation policy** as the policy type and click **Next**.
4. Enter a name for the policy. In our example, we named our policy Cluster ME Ext Cluster Sfw.
5. Click **OK**.

Figure 9-31 on page 352 shows the resulting panel.

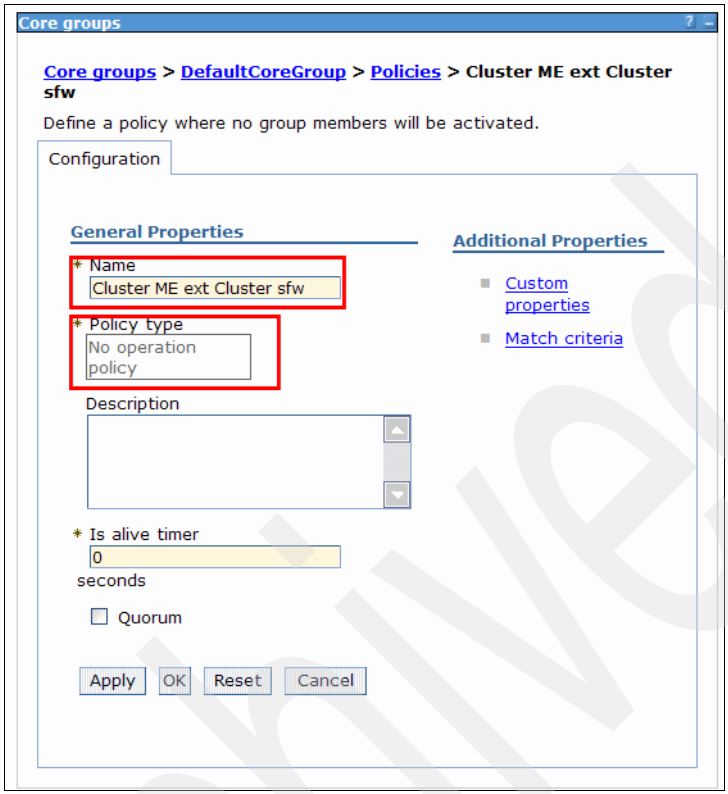


Figure 9-31 Create HA group policy for ME group

Next click **Match criteria** in the Additional Properties section. This allows you to set the match criteria needed to associate this policy with an HA group. Specifically, we want to match the HA group for the messaging engines. If you refer back to the paragraph above on the naming convention for the messaging engine HA groups, you can see that there are four parts to the name: WSAF_SIB_BUS, WSAF_SIB_MESSAGING_ENGINE, IBM_hc and type.

What we want to do is to define *one* policy that matches *all* messaging engine HA groups in the cluster. Therefore, we do not include the WSAF_SIB_MESSAGING_ENGINE part of the messaging engine HA group name because it has a value that is specific for each cluster member. If we were to include WSAF_SIB_MESSAGING_ENGINE as part of the match criteria, then we would have to define a NoOP policy for *each* cluster member. If we however do not include WSAF_SIB_MESSAGING_ENGINE, then we can define one policy and have it match all the messaging engine HA groups in the cluster.

In our example, there are already policies defined for the messaging engines. These are One of N policies for each messaging engine. We want to implement a single NoOP policy that overrides these existing policies. Notice in Figure 9-32 that the match criteria for the existing policies include two values as match criteria: the WSAF_SIB_MESSAGING_ENGINE and the type.

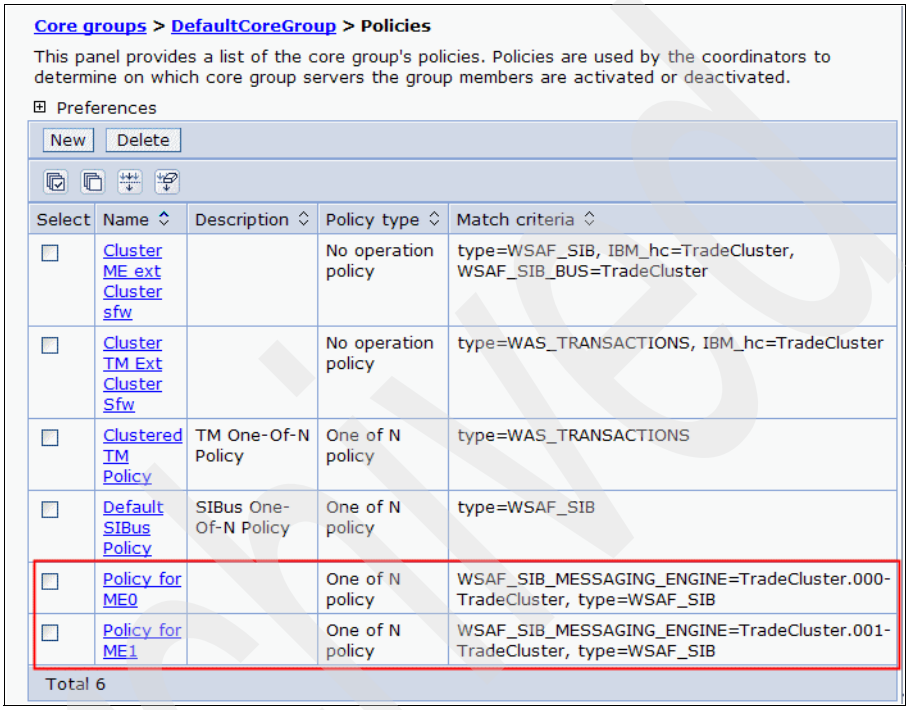


Figure 9-32 Existing HA policies for messaging engines

In order to override the existing policies in a definitive way, we need to define three values in the match criteria. By using three values, we have a more precise match criteria than the two values used by the existing policies. Therefore, we include the WSAF_SIB_BUS, type, and the IBM_hc values as match criteria for the new No Operation policy. For our example, the match criteria to be defined are `WSAF_SIB_BUS=TradeCluster`, `type=WSAF_SIB`, and `IBM_hs=TradeCluster`.

To do this:

1. Click **New**.
2. Enter `WSAF_SIB_BUS` as the Name and `TradeCluster` as the Value. Click **OK**.
3. Create the second match criteria by clicking **New** again.
4. Enter `type` as the Name and `WSAF_SIB` as the Value. Click **OK**.

5. Create the third match criteria by clicking **New** again.
6. Enter `IBM_hs` as the Name and `TradeCluster` as the Value. Click **OK**.

Figure 9-33 shows the result.

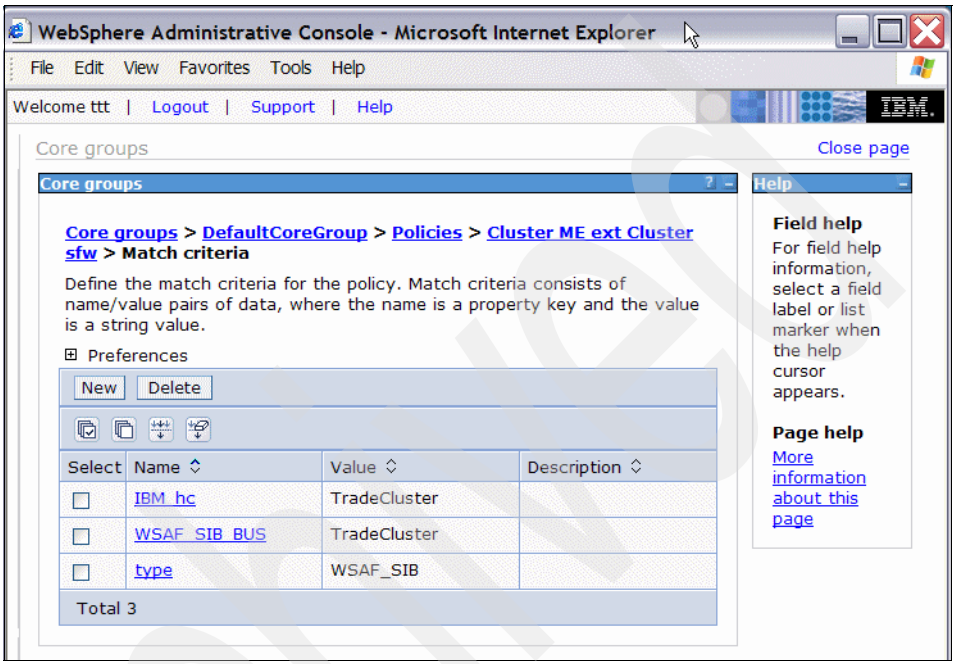


Figure 9-33 Match criteria for ME NoOP policy

Save your changes and make sure your configuration is synchronized to all nodes.

Now that we have WebSphere configured for the messaging engines No Operations policy, we must configure the external clustering software to control the operations of the failover.

9.7.4 Configuring external clustering software for default messaging provider No Operation policy

This section discusses how to configure external clustering software for the messaging engine No Operation policy. We do not describe any specific clustering software but discuss this in a generic way so it is valid for all clustering software. The remaining chapters of this part of the book discuss how to configure the specific clustering software (TSA, VERITAS Cluster Server, Sun Cluster).

Terminology

In order to discuss generically how to configure clustering software for the messaging engine NoOP policy, we first need an understanding of the common set of clustering software components. Refer to “Terminology” on page 355 for a description of these components.

Clustering software tasks

When the messaging engine is configured with NoOP policy, several functions must be coordinated with external clustering software. These functions are:

1. Starting the cluster members and then activating the ME in the cluster members that hosts the MEs.
2. Monitoring a specific cluster member’s ME and when it becomes unavailable, moving any dependent resources to another system and then notifying the cluster member on that system that it can start the ME. The HAMonitorWeb application is can also be used for monitoring the application servers in this scenario. See “HAMonitorWeb application” on page 330 for information.

In order to achieve these functions, the external clustering software must be configured with the following units of work, resources, and dependencies:

1. Start Server failover unit

   This failover unit is responsible for ensuring that the application server starts up and stays up on each system in the cluster. This failover unit is defined as an Active/Active failover unit, which means that the failover units start the servers up simultaneously on each system in the cluster and that there is no failover from one system to another when a server is down. If the monitoring programs notice that a critical resource in not available, an attempt is made to restart it on the system that is defined to host the resource. This failover unit consists of two resources:

   a. Start, stop, and monitor the Node Agent. This resource is defined as an application resource. The start and stop programs are invocations of the WebSphere **`startNode`** and **`stopNode`** commands. The monitoring is done by either specifying the process ID that needs to be active or by monitoring a port that the Node Agent listens on to ensure that it is listening. The specific monitoring technique used depends on the functionality provided by the external clustering software and is thus described in the individual chapter for the cluster software. This resource starts, stops, and monitors the Node Agents on each system in the cluster. Typically, this resource is not deemed as critical to this failover unit, which means that the Node Agent can be unavailable and the clustering software does not attempt to restart it. This is because the cluster members can still process runtime requests without the Node Agent being active.

b. Start, stop, and monitor the cluster member. This resource is defined as an application resource. The objective of this resource is to initiate the starting of the cluster members. This resource invokes the **startServer** and **stopServer** commands. This resource has a dependency on the Node Agent resource which must have completed its starting before this resource can start.

2. ME for Cluster Member1 failover unit

   This failover unit cannot start until the Start Server failover unit has completed its start. This failover unit is defined as a Active/Passive or Primary/Backup resource. This means that it is only active on a single system at one time. When the monitoring determines that a resource is unavailable, then the failover unit is failed over to the Standby/Backup system. The primary system for this failover unit is the system where Cluster Member1 is hosted. This failover unit has a single resource in it:

   a. Activate, deactivate, and monitor the ME for Cluster Member1. This resource is defined as an application resource. This resource uses the wasctrl-me script to perform its functions. It has different invocation strings registered for the primary and the secondary systems. The primary system is the system hosting the messaging engine for the cluster member (in this case Cluster Member1). The secondary system is the system hosting the Cluster Member2. This resource performs the following functions:

      i. It activates the messaging engine for the cluster member. The activation occurs on the primary cluster member initially and when the monitoring program determines that the messaging engine is unavailable there, it activates the messaging engine on the secondary cluster member.

      ii. It deactivates the messaging engine.

      iii. It monitors the status of the messaging engine.

3. ME for Cluster Member2 failover unit

   This failover unit cannot start until the Start Server failover unit has completed its start. This failover unit is defined as a Active/Passive or Primary/Backup resource. This means that it is only active on a single system at one time. When the monitoring determines that a resource is unavailable, then the failover unit is failed over to the Standby/Backup system. The primary system for this failover unit is the system where Cluster Member2 is hosted. This failover unit has a single resource in it:

   a. Activate, deactivate, and monitor the ME for Cluster Member2. This resource is defined as an application resource. This resource uses the wasctrl-me script to perform its functions. It has different invocation strings registered for the primary and the secondary systems. The primary system is the system hosting the messaging engine for the cluster member (in this

case Cluster Member2). The secondary system is the system hosting the Cluster Member1. This resource performs the following functions:

i. It activates the messaging engine for the cluster member. The activation occurs on the primary cluster member initially and when the monitoring program determines that the messaging engine is unavailable there, it activates the messaging engine on the secondary cluster member.

ii. It deactivates the messaging engine.

iii. It monitors the status of the messaging engine.

The chapters that follow provide examples of messaging engine No Operation policy configurations for a specific set of clustering software products.

Scripts to start, stop, and monitor WebSphere resources

There are five scripts available to start, stop, and monitor WebSphere resources for this scenario. They are called wasctrl-me, controlME.pty, activategroupME.pl, deactivategroupME.pl and monitorgroupME.pl. For additional information about the scripts, refer to the tables that follow. The TM NoOP scripts are comparable to these ME NoOP scripts.

Example 9-10 shows the wasctrl-me script.

Example 9-10 wasctrl-me

```
#!/bin/ksh
############################################################
#
# was ME service automation control script
#
# Input:
#       $1      action (<start|stop|status>)
#       $2      na_home, the nodeagent's home directory
#               USER_INSTALL_ROOT
#       $3      as_tcpp, the applicationServer's listening port
(WC_defaultHost)
#       $4      host_name, name of the system that is hosting the application
server
#       $5      host_port, SOAP_CONNECTOR_ADDRESS port of the application
server
#       $6      matchingCriteria, HA group matching criteria (for example
GN_PS=WASCell04\bottechiaNode01\TradeServer1,IBM_hc=myCluster,type=WAS_TRANSACT
IONS)
#       $7      script_path, directory that contains the scripts (Perl scripts
and tmHA.pty)
#
#
```

```
# History:
#
#   7.04.2005  initial version                          ms
#
#############################################################
#
# init section
#

UNKNOWN=0
ONLINE=110
OFFLINE=100

Action=${1:-status}
NA_HOME=$2
AS_TCPP=$3
HOST_NAME=$4
HOST_PORT=$5
CRITERIA=$6
SCRIPT_PATH=$7

export
PATH=$PATH:/bin:/usr/bin:/sbin:/usr/sbin:/usr/sbin/rsct/bin:${NA_HOME}/bin

#
#



case ${Action} in
        start)
               AS_UP=`netstat -lnt | grep :${AS_TCPP}`
                if [ "${AS_UP}" != "" ]; then
                   print  "Application Server is up so run the perl script"
                    RetC=`perl $SCRIPT_PATH/activategroupME.pl  ${AS_TCPP}
${CRITERIA}|awk '{print $1}'`
                   # rcs must be tested!!
                   if [ $RetC == "<OK/>" ]; then
                     RC=0
                   else
                     RC=42
                   fi
                   print "Transaction manager start rc: ${RC}"
                else
                   print "Application Server is down so run the wsadmin
command"
                  wsadmin.sh -conntype SOAP -host $HOST_NAME -port $HOST_PORT
-lang jython -f $SCRIPT_PATH/controlME.pty activate ${CRITERIA}
                 fi
```

```
                ;;
        stop)
                RetC=`perl $SCRIPT_PATH/deactivategroupME.pl  ${AS_TCPP}
${CRITERIA} |awk '{print $1}'`
               # rcs must be tested!!
                if [ $RetC == "<OK/>" ]; then
                  RC=0
                else
                  RC=42
                fi
                print "Transaction manager stop rc: ${RC}"
                ;;
        status)
                print "Requesting status"
                RetC=`perl $SCRIPT_PATH/monitorgroupME.pl  ${AS_TCPP}
${CRITERIA} `
                # rcs must be tested!!
                print "Status = ${RetC}"
                if [ $RetC == "<ACTIVE/>" ]; then
                  RC=$ONLINE
                else
                  RC=$OFFLINE
                fi

                print "Transaction manager status rc: ${RC}"
                ;;
        *)
                print "Error: Incorrect parameter >${Action}<"
                RC=${UNKNOWN}
                ;;
esac

exit ${RC}
```

Support scripts

Example 9-11 shows the controlME.pty script.

Example 9-11 controlME.pty

```
# This jython script is used to activate a HA service being managed by an HA
cluster. The HA cluster decides which machine the HA service should be
activated on and runs a script on that machine. The script invoked by the HA
cluster should be a shell script (bat file) that runs wsadmin targetting the
server that it wants the HA service to run on. This is achieved by setting the
host and port in the wsadmin conntype. The shell script (bat file) therefore
needs to be specialised on each node in the HA cluster.
```

```
# For example: Two machines in a cluster each have a shell script called
"controlscript"  and each wants to activate the messaging engine HA service
# # <-------------- "controlscript" ------------>
# machine 1: invoke wsadmin on server1's host and port : wsadmin -conntype SOAP
-host mach1 -port 8880 -lang jython controlHA.pty activate
IBM_hc=myCluster,type=WSAF_SIB
# The controlHA will activate <meName> in the server to which wsadmin is
connected

# machine 2: invoke wsadmin on server2's host and port: wsadmin -conntype SOAP
-host mach2 -port 8881 -lang jython controlHA.pty activate
IBM_hc=myCluster,type=WSAF_SIB
# The controlHA script will activate <meName> in the server to which wsadmin is
connected

#The controlHA script is passed the name of the cluster and the group type
"WSAF_SIB" to indicate messaging engine and starts the ME in the server process
to which wsadmin is connected. It does this by finding the local HAManager
MBean and calling the relevant operation on the MBean, e.g. for action==start,
it calls activateMember()

from com.ibm.ws.hamanager import AttributeNamesMemberProps
# Print usage information
def printHelp():
       print "Usage: controlHA <action> <matchingcriteria> where action is one
of {start|stop|monitor} and matchingcriteria is the HA policy matching criteria
to perform the action against (for example
GN_PS=WASCell04\bottechiaNode01\TradeServer1,IBM_hc=myCluster,type=WSAF_SIB)"
        return

# Get the HAManager MBean for the specified server.
def getServerHAMProxy(servername):
        strObjectName =
AdminControl.queryNames("type=HAManager,process="+servername+",*")
        objectName = AdminControl.makeObjectName(strObjectName)
        mb = TypedProxy.makeProxy(AdminControl, objectName,
"com.ibm.websphere.hamanager.jmx.CoordinatorJMX")
        return mb

# Find the group for the specified service and the status of its members
def getHAGroup(matchingCrit):
        global gHAM
#  matchSet="GN_PS="+cell+"\\"+node+"\\"+server+","+matchingCrit
#        print matchSet
        gs={}
        gs = gHAM.queryGroupState(matchingCrit,1,1)

   #Should be only one HAGroup that matches.
   if (len(gs) < 1):
```

```
            print "There is no HAGroup for "+matchingCrit
            return {}
        elif (len(gs) > 1):
           print "There are multiple HAGroups for  "+matchingCrit

           return {}
   else:
           # There is one HAGroup for the messaging engine
           groupName = gs[0].getGroupName()
           #displayGroupProperties(groupName) # optional information
           #displayGroupState(gs[0].getMemberData()) # optional information
           return groupName

#
# START
#

global gHAM
if (len(sys.argv) ==2):
        action = sys.argv[0]
        matchingCrit = sys.argv[1]

else:
        printHelp()
        sys.exit(1)

# Find out which cell and node and server we are running in.
node = AdminControl.getNode()
cell = AdminControl.getCell()

print "node is "+node
print "cell is "+cell
serverMBean = AdminControl.queryNames('processType=ManagedProcess,*')
server = AdminControl.getAttribute(serverMBean,'name')

print "server is "+server

# Connect to the HAM MBean in the specified server
# If that server is not running then the operation won't work anyway so give up
gHAM = {}
gHAM = getServerHAMProxy(server)
if (gHAM == {}):
   print "Cannot connect to HAManager MBean, giving up"
   sys.exit(1)

# Locate the HAGroup for the service
haGroup = {}
haGroup = getHAGroup(matchingCrit)
if (haGroup == {}):
```

```
        print "Cannot identify HAGroup for  "+matchingCrit+": giving up"
   sys.exit(1)

if (action == 'activate'):
        print "Attempting to activate " +matchingCrit+" on server
"+node+"."+server
        # Activate the local member
        gHAM.activateMember(haGroup,node,server)
elif (action == 'deactivate'):
        print "Attempting to deactivate "+matchingCrit+" on server
"+node+"."+server
        # Activate the local member
        gHAM.deactivateMember(haGroup,node,server)
elif (action == 'monitor'):
        print "Monitor action not yet implemented!!!"
else:
        printHelp()
        sys.exit(1)
```

Example 9-12 shows the activategroupME.pl script.

Example 9-12 activategroupME.pl

```
#
# Invocation:
# perl activategroup.pl cellname nodename clustername servername [portnumber]
#     where:
#             portnumber is the port number of the application that is doing
the activation.
#             matching is the HA matching criteria for the policy (for example
GN_PS=WASCell04\bottechiaNode01\TradeServer1,IBM_hc=myCluster,type=WAS_TRANSACT
IONS)
#

require HTTP::Request;
require LWP::UserAgent;
use Sys::Hostname;

if (@ARGV != 2) {
   print "Illegal number of arguments";
} else {
  $PORTNAME = $ARGV[0];
  $MATCHING = $ARGV[1]
}

my $hostname = hostname();
```

```
my $getrequest =
'http://'.$hostname.":".$PORTNAME."/HAMonitorWeb/ActivateLocalGroupMember?ms=".
$MATCHING."&s=a";

$request = HTTP::Request->new(GET => $getrequest);

$ua = LWP::UserAgent->new();
$response = $ua->request($request);
if ($response->is_success) {
  $result = $response->content;
  if(index($result, "<OK/>", 0) >= 0)
  {
    print "<OK/>";
  }
  else
  {
    print "<NOT OK/>";
  }
}
else
{
  print "1";
  print STDERR $response->status_line, "\n";
}
```

Example 9-13 shows the deactivategroupME.pl script.

Example 9-13 deactivategroupME.pl

```
#
# Invocation:
# perl deactivategroup.pl cellname nodename clustername servername [portnumber]
#     where:
#             cellname is the name of the cell that contains the application
server or cluster member
#             nodename is the name of the node that contains the application
server or cluster member
#             servername is the name of the server or cluster member that
requires the group to be activated.
#             portnumber is the port number of the application that is doing
the activation.
#            matching is the HA matching criteria for the policy (for example
GN_PS=WASCell04\bottechiaNode01\TradeServer1,IBM_hc=myCluster,type=WAS_TRANSACT
IONS)
#


require HTTP::Request;
require LWP::UserAgent;
```

```
use Sys::Hostname;

if (@ARGV != 2) {
   print "Illegal number of arguments";
} else {
  $PORTNUMBER = $ARGV[0];
  $MATCHING = $ARGV[1]
}

my $hostname = hostname();
my $getrequest =
'http://'.$hostname.":".$PORTNUMBER."/HAMonitorWeb/ActivateLocalGroupMember?ms=
".$MATCHING."&s=d";

$request = HTTP::Request->new(GET => $getrequest);

$ua = LWP::UserAgent->new();
$response = $ua->request($request);
if ($response->is_success) {
  $result = $response->content;
  if(index($result, "<OK/>", 0) >= 0)
  {
    print "<OK/>";
  }
  else
  {
    print "<NOT OK/>";
  }
print $result;
}
else
{
  print "1";
  print STDERR $response->status_line, "\n";
}
```

Example 9-14 shows the monitorgroupME.pl script.

Example 9-14 monitorgroupME.pl

```
#
# Invocation:
# perl monitorgroup.pl cellname nodename clustername servername [portnumber]
#     where:
#              cellname is the name of the cell that contains the application
server or cluster member
#              nodename is the name of the node that contains the application
server or cluster member
```

```
#              servername is the name of the server or cluster member that
requires the group to be activated.
#              portnumber is the port number of the application that is doing
the activation.
#             matching is the HA matching criteria for the policy (for example
IBM_hc=myCluster,type=WAS_TRANSACTIONS)
#


require HTTP::Request;
require LWP::UserAgent;
use Sys::Hostname;

if (@ARGV != 5) {
   print "Illegal number of arguments";
} else {
  $CELLNAME= $ARGV[0];
  $NODENAME = $ARGV[1];
  $SERVERNAME = $ARGV[2];
  $PORTNUMBER = $ARGV[3];
  $MATCHING = $ARGV[4]
}

my $hostname = hostname();
my $getrequest =
'http://'.$hostname.":".$PORTNUMBER."/HAMonitorWeb/QueryLocalMemberState?ms=GN_
PS=".$CELLNAME."\\".$NODENAME."\\".$SERVERNAME.",".$MATCHING."&s=a";

$request = HTTP::Request->new(GET => $getrequest);

$ua = LWP::UserAgent->new();
$response = $ua->request($request);
if ($response->is_success) {
  $result = $response->content;
  if(index($result, "<ACTIVE/>", 0) >= 0)
  {
    print "<ACTIVE/>";
  }
  else
  {
    print "<INACTIVE/>";
  }
}
else
{
  print "1";
  print STDERR $response->status_line, "\n";
}
```

10

WebSphere and IBM Tivoli System Automation

This chapter provides an introduction into the basics of clustering IBM WebSphere Application Server V6 using Tivoli System Automation.

Important: At the time of writing this redbook, Tivoli System Automation was not in the list of supported software. However, the following configurations exist:

- Supported
- Not supported
- Other

Tivoli System Automation falls into the *Other* category. For details about how IBM Support handles this category, see the document *IBM WebSphere Application Server - Clarification of configurations support* that is available at:

http://www.ibm.com/support/docview.wss?rs=180&context=SSEQTP&uid=swg27004311

For the latest list of supported products see:

http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html

10.1 Introduction to Tivoli System Automation

IBM Tivoli System Automation for Multiplatforms is a product that provides high availability (HA) by automating the control of IT resources such as processes, file systems, IP addresses and other resources. It facilitates the automatic switching of users, applications, and data from one system to another in the cluster after a hardware or software failure.

10.1.1 How Tivoli System Automation works

The IBM Tivoli System Automation product provides HA by automating resources such as processes, applications, and IP addresses. To automate an IT resource (for example, an IP address), the resource must be defined to Tivoli System Automation. Every application must be defined as a resource in order to be managed and automated with Tivoli System Automation. Application resources are usually defined in the generic resource class IBM.Application. For the HA IP address, the resource class IBM.ServiceIP must be used.

Reliable Scalable Cluster Technology, or RSCT, is a product fully integrated into IBM Tivoli System Automation. RSCT is a set of software products that together provide a comprehensive clustering environment for AIX and Linux. RSCT is the infrastructure to provide clusters with improved system availability, scalability, and ease of use. RSCT provides three basic components, or layers, of functionality:

- RMC (Resource Monitoring and Control), provides global access for configuring, monitoring, and controlling resources in a peer domain.
- HAGS (High Availability Group Services), is a distributed coordination, messaging, and synchronization service.
- HATS (High Availability Topology Services), provides a scalable heartbeat for adapter and node failure detection, and a reliable messaging service in a peer domain.

Terminology

Some of the key terms used in describing Tivoli System Automation are:

- Cluster or peer domain

  The group of host systems upon which Tivoli System Automation manages resources is known as a cluster. A cluster can consist of one or more systems or nodes.

- Resource

  A resource is any piece of hardware or software that can be defined to IBM Tivoli System Automation. These resources can be either defined manually by the administrator using the **mkrsrc** (make resource) command or through

the “harvesting” functionality of the cluster infrastructure, whereby resources are automatically detected and prepared for use. All resources are controlled through the appropriate resource managers.

- Resource class

  A resource class is a collection of resources of the same type. For example, if an application is a resource, then all applications defined in the cluster would comprise a resource class. Resource classes allow you to define the common characteristics among the resources in its class. In the case of applications, the resource class can define identifying characteristics, such as the name of the application, and varying characteristics, such as whether or not the application is running. So each resource in the class can then be noted by its characteristics at any given time.

- Resource group

  Resource groups are logical containers for a collection of resources. This container allows you to control multiple resources as a single logical entity. Resource groups are the primary mechanism for operations within IBM Tivoli System Automation. Resource groups can also be nested, meaning that applications can be split into several resource groups which themselves are part of another higher level resource group. Also, resource groups can be defined in such a way that their members can be located on different systems in the cluster.

- Managed resource

  A managed resource is a resource that has been defined to IBM Tivoli System Automation. To accomplish this, the resource is added to a resource group, at which time it becomes manageable through Tivoli System Automation.

- Nominal state

  The nominal state of a resource group indicates to Tivoli System Automation whether the resources with the group should be Online or Offline at this point in time. So setting the nominal state to .Offline indicates that you wish for Tivoli System Automation to stop the resources in the group, and setting the nominal state to .Online. is an indication that you wish to start the resources in the resource group.

- Equivalency

  An equivalency is a collection of resources that provides the same functionality. For example, equivalencies are used for selecting network adapters that should host an IP address. If one network adapter goes offline, Tivoli System Automation selects another network adapter to host the IP address.

- Relationships

  IBM Tivoli System Automation allows for the definition of relationships between resources in a cluster. There are two different relationship types:

  - Start/stop relationships

    These relationships are used to define start and stop dependencies between resources. You can use the StartAfter, StopAfter, DependsOn, DependsOnAny, and ForcedDownBy relationships to achieve this. For example, a resource must only be started after another resource was started. You can define this by using the policy element StartAfter relationship.

  - Location relationships

    Location relationships are applied when resources must, or should if possible, be started on the same or a different node in the cluster.

- Resource manager

  Resource classes are managed by the various resource managers (RM), depending on what type of resource is being managed. A resource manager is a software layer between a resource and RMC. The following resource managers are provided by IBM Tivoli System Automation:

  - Recovery RM (IBM.RecoveryRM)

    This resource manager serves as the decision engine for IBM Tivoli System Automation. When a policy for defining resource availabilities and relationships is defined, this information is supplied to the Recovery RM. This RM runs on every node in the cluster, with exactly one Recovery RM designated as the master. The master evaluates the monitoring information from the various resource managers. When a situation develops that requires intervention, the Recovery RM drives the decisions that result in start or stop operations on the resources as needed.

  - Global Resource RM

    The Global Resource RM (IBM.GblResRM) supports two resource classes:

    - IBM.Application

      The IBM.Application resource class defines the behavior for general application resources. This class can be used to start, stop, and monitor processes. As a generic class, it is very flexible and can be used to monitor and control various kinds of resources. Most of the applications that you automate are done using this class.

- IBM.ServiceIP

  This application class defines the behavior of Internet Protocol (IP) address resources. It allows you to assign IP addresses to an adapter. In effect, it allows IP addresses to “float” among nodes.

– Configuration RM

  The Configuration RM (IBM.ConfigRM) is used in the cluster definition. In addition, quorum support, which is a means of insuring data integrity when portions of a cluster lose communication, is provided.

– Event Response RM

  The Event Response RM (IBM.ERRM) provides the ability to monitor conditions in the cluster in order for the RMC system to react in certain ways.

– Test RM

  The Test resource manager (IBM.TestRM) manages test resources and provides functions to manipulate the operational state of these resources. The resource manager is operational in a peer domain mode only and provides the resource class IBM.Test.

For more detailed information about Tivoli System Automation Resource Managers see *IBM Tivoli System Automation for Multiplatforms Guide and Reference* found at:

http://publib.boulder.ibm.com/tividd/td/ITSAFL/SC33-8210-03/en_US/PDF/halgre11.pdf

10.1.2 Configuration basics of Tivoli System Automation

Configuring Tivoli System Automation to automate or to manage resources involves the following basic steps:

- Creating a Tivoli System Automation Domain
- Creating a resource group
- Creating resources
- Adding resources to resource group
- Creating equivalencies (typically used for IP address resources)
- Specifying dependencies

In order to create a new resource of the Application resource class (for example, for the Deployment Manager), the following three scripts (or commands respectively) must be provided:

1. A start script (or command) to bring the resource online.
2. A stop script (or command) to take the resource offline.
3. A script (or command) to monitor the resource through polling.

For details on how to develop these scripts, refer to *IBM Tivoli System Automation for Multiplatforms Guide and Reference.*

10.1.3 Managing resources

When creating resources, two approaches have proven to work:

- Create resources from definitions.

  Have definition files with the attributes for each resource in classes such as IBM.Application, IBM.ServiceIP. Those are referenced by a simple script that creates the resources using the RSCT **`mkrsrc -f`** command.

  The advantages are that you learn how to create resources using RSCT commands and you will be able to change the script easily. You can mix and match definitions and scripts.

  However, one of the chief drawbacks is that if you want to, for example, add a node, you must change the *same* NodeNameList parameter in each definition file.

- Create resources from a configuration file.

  Have a single configuration file that feeds a script to make the resources. The script must be sophisticated enough to assemble the **`mkrsrc`** command arguments from the configuration dynamically.

  The advantage here is that you define variables in simple syntax only *once* in the configurations file, while the script interprets it and generates the various **`mkrsrc`** calls on the fly. However, you will not learn about the **`mkrsrc`** command and the attributes to make resources for other applications. The *smart* script is more difficult to understand and to change.

  The recommendation is:

  – If the automation structure in terms of automated resources and their relationships will not change, use the configuration file approach.

  – If it is likely that you must extend the automated resources and add relationships, use the definitions approach.

  To benefit from both, we combine these approaches by providing a configuration file that can be tailored. Executing the cfg-script generates .def files and the corresponding script to make resources. If option -p is specified, the resources are created directly.

  Note, that this is also useful in three other aspects:

  a. Studying the make script, you learn the commands to remove and generate automated resources.

  b. If **–p** is not specified, then you can study the .def files to learn how the Tivoli System Automation definitions for the attributes look. That way,

when you list resource attributes using Tivoli System Automation commands, you will be able to recognize the resources you created easily.

c. You can check what resources will be removed and created without really making them, and can read just the configuration and repeatedly run the configuration script until you are satisfied with the resources that will be created.

Note that neither approach has direct support for removing resources selectively. A remove command must be used similar to what is generated in the make-script; this remove command must then be executed separately.

10.1.4 Tivoli System Automation and IBM WebSphere MQ

Currently, there are no plans to make a Cluster Agent or sample scripts available to use IBM WebSphere MQ with Tivoli System Automation.

10.1.5 Using Cluster Agent for IBM DB2 UDB

Information about how to make DB2 highly available using Tivoli System Automation can be downloaded from:

`http://www.ibm.com/software/tivoli/products/sys-auto-linux/downloads.html`

Download the document entitled *Highly available DB2 with Tivoli System Automation for Linux* as well as the *Sample policies and scripts to make DB2 highly available*.

10.2 Planning and preparation

Before proceeding to configure Tivoli System Automation for WebSphere, ensure the following:

- Install IBM Tivoli System Automation for Multiplatforms 2.1 following the instructions in the *IBM Tivoli System Automation for Multiplatforms Guide and Reference*.
- WebSphere must *not* be installed to use the host names of either server. Instead, a third virtual/alias IP address should be obtained along with a host name for that Virtual IP Address. The Virtual IP Address is a separate IP address in the cluster and does not match any IP address assigned to the network adapters on each cluster node, that are made in system definitions outside of Tivoli System Automation. This address is in contrast created by Tivoli System Automation and is an additional alias address on an appropriate network adapter on the node where the Deployment Manager resides. When the Deployment Manager moves to a new location, the alias

address is removed from the former node and recreated on the new node, where the Deployment Manager is about to be restarted.

10.3 Deployment Manager

This section discusses the steps needed to make the Deployment Manager highly available using Tivoli System Automation.

Scenario description

The environment for the Deployment Manager failover scenario is as follows:

- Two nodes in the peer domain, one production node and one standby node.
- A file system containing the WebSphere files that are shared between the production node and the standby node.
- Cold failover to the standby node in case of, for example, a failure of the production node's operating system, network adapter, or other hardware components.

Resources

- One Deployment Manager
- One IP address

The resources are of type floating.

Network equivalencies

An equivalency is defined for the network interfaces on which the floating IP address depends.

Resource groups

The Deployment Manager and IP address are members of the same resource group.

Relationships between resources

- The Deployment Manager depends on the IP address.
- The IP address depends on the network interface equivalency.

Hardware topology

The hardware used in our tests consisted of two blade servers called thost1 and thost2 running in an IBM Blade Center. Both blades have a fiber adapter installed and a LUN with the shared disk is available to both blades. The LUN is managed by an IBM TotalStorage SAN Volume Controller (SVC) which uses a FAStT900 for its storage. The FAStT900 has a large block of storage that is managed by

the SVC. The SVC is configured to have a VLUN of around 5GB for use by our Deployment Manager. That VLUN is mapped so that it is visible on both thost1 and thost2.

10.3.1 Installing the Deployment Manager

Install and configure the WebSphere components (here the WebSphere binaries and the Deployment Manager profile) on a shared file system concurrently mounted on both servers. A shared disk that is physically attached to both the primary and backup servers but without the file system mounted on both servers can also be used. However, that requires that Tivoli System Automation manages the file system failover.

In our tests, we used a shared file system. A shared file system could be an NFS file system or a Samba file system.

Note: If NFS V3 or V2 is used, it is possible that in case of a failure of the primary system the administrator might need to clear file locks on the NFS volume manually prior to failover.

This is not a problem with NFS V4 or Windows shared file systems as the locks are automatically released. To determine whether your file system clears locks, you can run the test that is discussed in 6.7.4, “File System Locking Protocol Test” on page 213.

The file server should be itself highly available and the disks that the file server uses should be RAID volumes that offer data protection in the event of disk failure. If this approach is used then there is no need for external software such as Tivoli System Automation to manage the file system.

For more details on how to install and configure the Deployment Manager refer to 9.3, “Deployment Manager high availability” on page 298.

10.3.2 Configuring Tivoli System Automation to run the Deployment Manager scenario

As mentioned in 10.1.2, “Configuration basics of Tivoli System Automation” on page 371, there are several steps involved when configuring Tivoli System Automation. This section explains these steps for the Deployment Manager scenario.

Creating and starting the Tivoli System Automation domain

We create a dedicated domain for this scenario. A domain is simply a Tivoli System Automation cluster. The domain consists of nodes or physical boxes on which applications or resources can be placed. First, prepare both servers:

1. Execute the following command on thost1:

   ```
   preprpnode thost1 thost2
   ```

2. Execute the following command on thost2:

   ```
   preprpnode thost1 thost2
   ```

3. Create the domain and execute the command on either server. We use thost2 in this example.

   ```
   mkrpdomain DMGRDOMAIN thost1 thost2
   ```

4. Start the domain:

   ```
   startrpdomain DMGRDOMAIN
   ```

Adding a tie-breaker disk for quorum purposes is recommended as a best practice and is necessary when the domain has an even number of nodes. For details on how to add a tie-breaker disk see *IBM Tivoli System Automation for Multiplatforms Guide and Reference* found at:

```
http://publib.boulder.ibm.com/tividd/td/ITSAFL/SC33-8210-03/en_US/PDF/
halgre11.pdf
```

Creating a resource group

We first make a resource group. A resource group is a container for a set of resources that should be operated on as a single entity. It does not matter on which system this and the following commands are executed.

```
mkrg dmgr-rg
```

Creating resources

Next, we need to create resource definition files for the Deployment Manager and the IP address to be managed by Tivoli System Automation as shown in Example 10-1 on page 377 and Example 10-2 on page 377. Using these examples, create your own definition text files (for example dmgr-jvm.def) and run the following commands to create the resources:

```
mkrsrc -f dmgr-ip.def IBM.ServiceIP
mkrsrc -f dmgr-jvm.def
```

Example 10-1 dmgr-jvm.def

```
PersistentResourceAttributes::
Name=dmgr-jvm
ResourceType=1
StartCommand=/mnt/sanfs/sanfs/test/tsa/nd/dm_start.ksh
StopCommand=/mnt/sanfs/sanfs/test/tsa/nd/dm_stop.ksh
MonitorCommand=/mnt/sanfs/sanfs/test/tsa/nd/dm_monitor.ksh
StartCommandTimeout=120
StopCommandTimeout=60
MonitorCommandTimeout=9
MonitorCommandPeriod=30
ProtectionMode=1
NodeNameList={'thost1','thost2'}
UserName=root
```

Example 10-2 dmgr-ip.def

```
PersistentResourceAttributes::
Name="dmgr-ip"
ResourceType=1
IPAddress=192.168.10.3
NetMask=255.255.255.0
ProtectionMode=1
NodeNameList={"thost1","thost2"}t
```

The scripts to start (dm_start.ksh), stop (dm_stop.ksh) and monitor (dm_monitor.ksh) the Deployment Manager are shown in Example 10-3, Example 10-4 on page 378, and Example 10-5 on page 378.

Example 10-3 dm_start.ksh

```
#!/bin/ksh -p
PATH=/bin:/usr/bin:/sbin:$PATH

logger -i -p info -t $0 "Starting deployment manager"
/mnt/sanfs/sanfs/test/was/profiles/dmgr/bin/startManager.sh
rc=$?
logger -i -p info -t $0 "DM started with exit code " $rc
return $rc
```

Example 10-4 dm_stop.ksh

```
#!/bin/ksh -p
PATH=/bin:/usr/bin:/sbin:$PATH
logger -i -p info $0 "Stopping Deployment Manager"
/mnt/sanfs/sanfs/test/was/profiles/dmgr/bin/stopManager.sh
logger -i -p info $0 "Stopped Deployment Manager"
return 0
```

Example 10-5 dm_monitor.ksh

```
#!/bin/ksh -p
PATH=/bin:/usr/bin:/sbin:$PATH
logger -i -p info $0 "Monitor Deployment Manager"
running=$(netstat -a | grep 9061 | wc -l)
if [ ${running} -eq 0 ]; then
   logger -i -p info "Deployment Manager not running"
   return 2
fi
logger -i -p info $0 "Deployment Manager running"
return 1
```

Important: The monitor script is executed periodically by Tivoli System Automation to check if the Deployment Manager is running. This script needs to be very efficient as it will be executed continually. The script can be written to use a variety of approaches to check if a Deployment Manager is running. The above example uses the **netstat** command to see if there is a process listening on the HTTP port for our Deployment Manager. The number 9061 in this script should be changed to be the HTTP port for your Deployment Manager. This is a basic test. Additional tests can be incorporated into this script for a more robust test. For example, an HTTP client could ping the home page of the console. If an HTML page is received, then the Deployment Manager is working.

The **wsadmin serverstatus** command should *never* be used to check if the Deployment Manager is running. It takes a lot of CPU when it starts, and it takes around 20 to 30 seconds to complete.

Adding resources to the resource group

Run the following commands to add the previously created resources to the resource group:

```
addrgmbr -g dmgr-rg IBM.ServiceIP:dmgr-ip
addrgmbr -g dmgr-rg IBM.Application:dmgr-jvm
```

Creating equivalencies

Create an equivalency for the network adapters on both servers that we want to use for the IP alias:

```
mkequ dmgr-ip-equ IBM.NetworkInterface:eth0:thost1,eth0:thost2
```

Specifying dependencies

Execute the following command to specify that the IP address is dependant on the equivalence:

```
mkdep -p DependsOn -S IBM.ServiceIP:dmgr-ip \
-G IBM.Equivalence:dmgr-ip-equ dmgr-ip-rel-equ
```

Execute the following command to specify that the Deployment Manager is dependant on the IP:

```
mkdep -p DependsOn -S IBM.Application:dmgr-jvm -G IBM.ServiceIP:dmgr-ip \
dmgr-jvm-rel-ip
```

10.3.3 Testing Deployment Manager failover

Now that the Deployment Manager is automated, we can bring it online by telling Tivoli System Automation to bring the resource group online. This is done using the following command on any node in the Tivoli System Automation domain:

```
chrg -o online dmgr-rg
```

After executing this command, you should see the Deployment Manager starting on one of the two nodes, either thost1 or thost2. You can bring the Deployment Manager down again using the command:

```
chrg -o offline dmgr-rg
```

Note: The Deployment Manager should never be started using the **startManager** command or by using a start script (for example, rc.d). It should only be started by Tivoli System Automation. This ensures that the IP address and any file systems (if managed by Tivoli System Automation) are also mounted correctly.

We can test failover by first bringing the Deployment Manager online using the **chrg** command and then telling Tivoli System Automation to remove the node where it's currently running from the allowed list. Tivoli System Automation keeps a list of nodes that are to be excluded from running a resource group. If the Deployment Manager was running on thost1, then the following command excludes thost1 from running the Deployment Manager. You should see the Deployment Manager being shut down on thost1 and then started on thost2:

```
samctrl -u a thost1
```

This adds thost1 to the excluded node list for the domain. This triggers a orderly failover to thost2. When the Deployment Manager is running on thost2 then you can undo this command using:

```
samctrl -u d thost1
```

This removes the node thost1 from the disabled node list and makes thost1 available again for running the Deployment Manager. No automatic failback occurs. You can fail the Deployment Manager back to thost1 by first excluding thost2 using **`samctrl -u a thost2`** and then, when the Deployment Manager is running, undo that command.

This is just one of the possible test scenarios. You can run other test scenarios, such as killing the Deployment Manager process, shutting down the system where the Deployment Manager runs or unplugging the network cable. All of these events should trigger a failover.

10.4 Node Agent and application server

This section demonstrates how to make a WebSphere Node Agent and related application server on a two-node cluster highly available. This scenario can be scaled up to N nodes by following the steps for two nodes.

Scenario description

- Two nodes in the peer domain, one production node and one standby node. A Node Agent and related application server run on the production node.
- A file system containing the WebSphere files that are shared between the production node and the standby node.
- Cold failover to the standby node in case of, for example, a failure of the production node's operating system, network adapter, or other hardware components.

Resources

- One Node Agent
- One application server
- One IP address

The resources are of type floating.

Network equivalencies

An equivalency is defined for the network interfaces on which the floating IP address depends.

Resource groups

The Node Agent, application server and IP address are members of the same resource group.

Relationships between resources

- The IP address depends on the network interface equivalency.
- The Node Agent depends on the IP address.
- The application server depends on the Node Agent.

Figure 10-1 illustrates the resources and dependencies for the Node Agent and application server failover scenario.

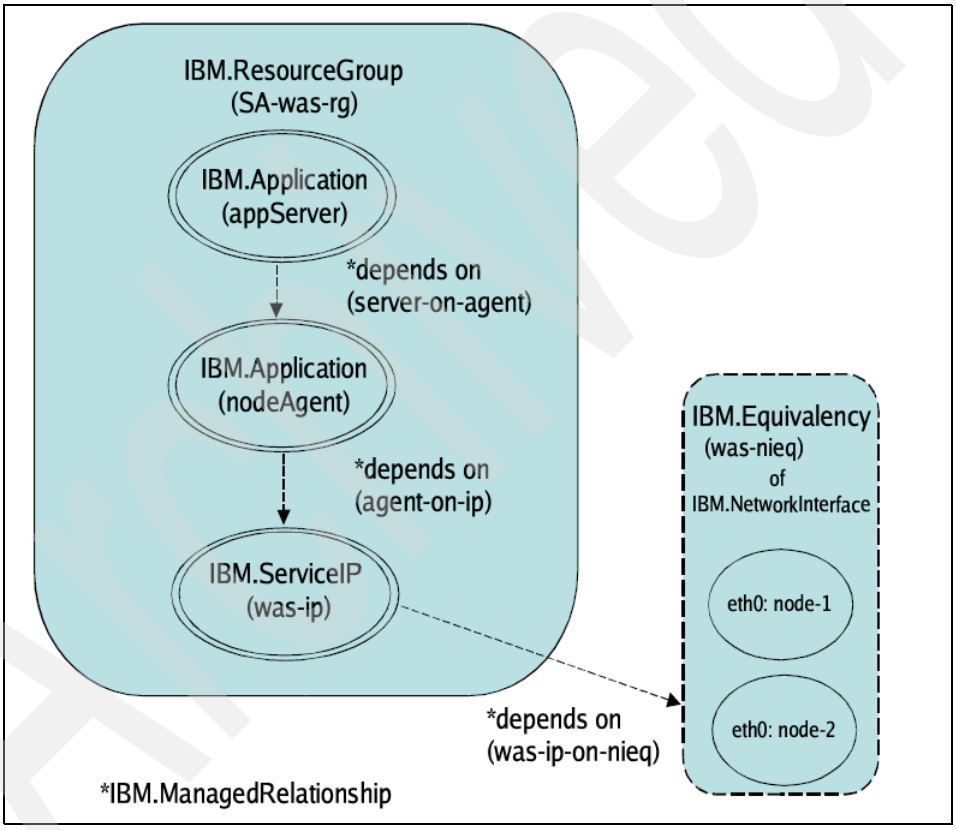


Figure 10-1 Node Agent and application server scenario- resources and dependencies

Hardware topology

The hardware used in our tests consisted of two blade servers called thost1 and thost2 running in an IBM Blade Center. Both blades have a fiber adapter installed and a LUN with the shared disk is available to both blades. The LUN is managed

by an IBM TotalStorage SAN Volume Controller (SVC) which uses a FAStT900 for its storage. The FAStT900 has a large block of storage that is managed by the SVC. The SVC is configured to have a VLUN of around 5GB for use by the Node Agent and application servers. That VLUN is mapped so that it is visible on both thost1 and thost2.

10.4.1 Installing a Node Agent and application server or servers

In our setup, we opted for simplicity and configured our system to run in cold failover mode (otherwise known as Active/Passive). The WebSphere binaries and profiles are located on a shared file system.

Figure 10-2 on page 383 shows our lab setup. We used two systems that have access to a Network Attached Storage (NAS) device. The systems use iSCSI to attach to the NAS and this provides simultaneous and concurrent access to the directories that contain the WebSphere configuration data. We configured Tivoli System Automation to have an Active/Passive configuration. This means the Node Agent and application servers are running only on one of the systems at a time. One system is deemed the primary system and when the processes fail on that system, the secondary system becomes active and all requests are served by the secondary system.

See 9.4, “Node Agent and application server high availability” on page 304 for details on how to install and configure WebSphere for this scenario.

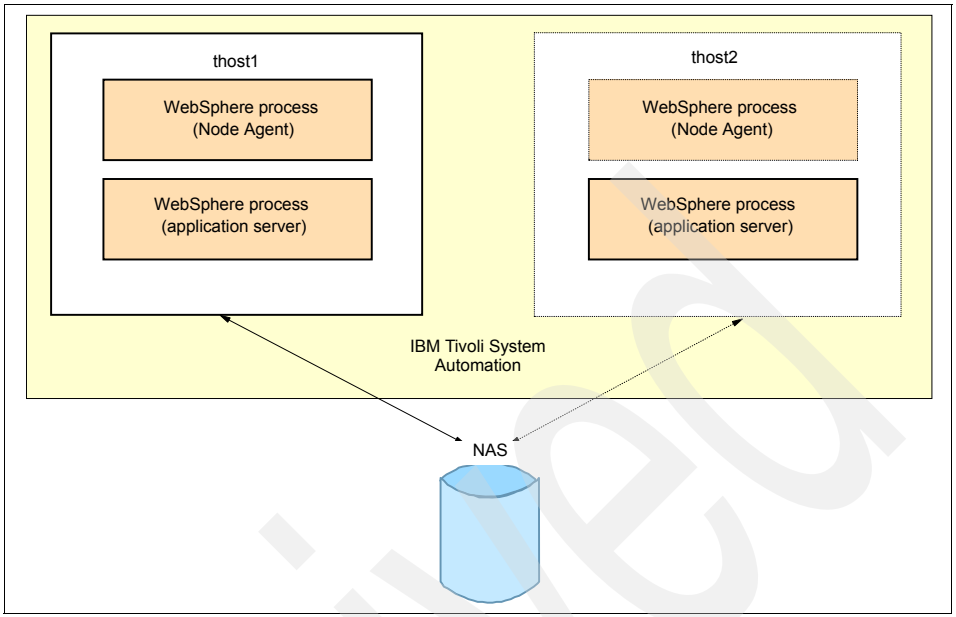


Figure 10-2 Tivoli System Automation: Node Agent and application server setup

10.4.2 Configuring Tivoli System Automation to run the Node Agents and application server

The Tivoli System Automation configuration steps are similar to those described for the Deployment Manager in 10.3.2, “Configuring Tivoli System Automation to run the Deployment Manager scenario” on page 375. In this section we provide example Tivoli System Automation commands, resource definitions, stop, start and monitor scripts, and describe in detail any additional configuration steps necessary to configure Tivoli System Automation to run the Node Agent and application server.

Creating and starting the Tivoli System Automation domain

First you need to run the **`preprpnode`** command on both nodes, then create a domain, start the domain and add the tie-breaker if needed as described in “Creating and starting the Tivoli System Automation domain” on page 376.

Completing the Tivoli System Automation configuration

Run the commands shown in Example 10-6 on page 384 to create resource groups, make resources, add resources to resource groups, make network equivalencies and to specify dependencies. When this is done, the system is

managed by Tivoli System Automation and all processes on the system need to be started/stopped using Tivoli System Automation.

Dependencies

It is important to understand the dependencies between the resources:

1. The IP address depends on network interface equivalency.
2. The Node Agent depends on the IP address.
3. The application server depends on the Node Agent.

This means that the network equivalency of the two boxes and the file system must be online before the Node Agent can be online. Also, the Node Agent must be online before the application server can be started.

Important: Due to the nature of IP, all of the hosts involved in the network equivalency must be on the same subnet for IP failover to work properly.

Example 10-6 Commands for Node Agent and application server HA

```
# make resource group(s)
mkrg SA-was-rg

# make resources
mkrsrc -f SA-was-na.def IBM.Application
mkrsrc -f SA-was-as.def IBM.Application
mkrsrc -f SA-was-ip-1.def IBM.ServiceIP

# make network equivalencies
mkequ was-node1-ip-equ IBM.NetworkInterface:eth0:thost2,eth0:thost1

# add resources to resource group
addrgmbr -m T -g SA-was-rg IBM.Application:SA-was-na
addrgmbr -m T -g SA-was-rg IBM.Application:SA-was-as
addrgmbr -m T -g SA-was-rg IBM.ServiceIP:SA-was-ip-1

# specify dependencies and equivalencies
mkrel -S IBM.Application:SA-was-na -G IBM.ServiceIP:SA-was-ip-1 -p DependsOn
SA-was-na-on-ip-1
mkrel -S IBM.ServiceIP:SA-was-ip-1 -G IBM.Equivalency:SA-was-nieq-1 -p
DependsOn SA-was-ip-on-nieq-1
mkrel -S IBM.Application:SA-was-as -G IBM.Application:SA-was-na -p DependsOn
SA-was-as-on-na
mkequ SA-was-nieq-1 IBM.NetworkInterface:eth0:thost1,eth0:thost2
```

Example resource definition files for a Virtual IP Address, Node Agent, and application server are shown in the following examples.

Example 10-7 SA-was-ip-1.def

```
PersistentResourceAttributes::
  Name="SA-was-ip-1"
  ResourceType=1
  IPAddress=192.168.4.1
  NetMask=255.255.255.0
  ProtectionMode=NodeNameList={"thost1","thost2"}
```

Example 10-8 SA-was-na.def

```
PersistentResourceAttributes::
Name=SA-was-na
ResourceType=1
StartCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-na start
/opt/IBM/WebSphere/AppServer/profiles/node1 8878
StopCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-na stop
/opt/IBM/WebSphere/AppServer/profiles/node1 8878
MonitorCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-na status
/opt/IBM/WebSphere/AppServer/profiles/node1 8878
StartCommandTimeout=60
StopCommandTimeout=60
MonitorCommandTimeout=19
MonitorCommandPeriod=30
ProtectionMode=1
RunCommandsSync=0
NodeNameList={'thost1','thost2'}
UserName=root
```

Example 10-9 SA-was-as.def

```
PersistentResourceAttributes::
Name=SA-was-as
ResourceType=1
StartCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-as start
/opt/IBM/WebSphere/AppServer/profiles/node1 8879 thost1
StopCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-as stop
/opt/IBM/WebSphere/AppServer/profiles/node1 8879 thost1server
MonitorCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-as status
/opt/IBM/WebSphere/AppServer/profiles/node1 8879 thost1server
StartCommandTimeout=180
StopCommandTimeout=180
MonitorCommandTimeout=19
MonitorCommandPeriod=30
ProtectionMode=1
RunCommandsSync=0
```

```
NodeNameList={'thost1','thost2'}
UserName=root
```

Example start, stop, and monitor scripts for the Node Agent and application server are shown in Example 10-10 and Example 10-11 on page 387. These scripts are also used in the subsequent scenarios in this chapter.

Example 10-10 wasctrl-na

```
#!/bin/ksh
############################################################
#
# was nodeAgent automation control script
#
# Input:
#       $1      action (<start|stop|status>)
#       $2      na_home, root directory of node configuration
#               USER_INSTALL_ROOT
#       $3      na_tcpp, node agent's SOAP_CONNECTOR_ADDRESS
#
#
############################################################
#
# init section
#

UNKNOWN=0
ONLINE=1
OFFLINE=2

Action=${1:-status}
NA_HOME=$2
NA_TCPP=$3

export
PATH=$PATH:/bin:/usr/bin:/sbin:/usr/sbin:/usr/sbin/rsct/bin:${NA_HOME}/bin


case ${Action} in
        start)
                startNode.sh -timeout 600
                RC=$?
                logger -i -p info -t $0 "NodeAgent start rc: ${RC}"
                ;;
        stop)
                stopNode.sh -timeout 180
                RC=$?
                if [ $RC -eq 0 ]; then
                  logger -i -p info -t $0 "NodeAgent stop rc: ${RC}"
```

```
                elif [ $RC -eq 246 ]; then
                  logger -i -p info -t $0 "NodeAgent stop rc: ${RC} (NodeAgent
did not run)."
                else
                  logger -i -p info -t $0 "NodeAgent stop rc: ${RC}"
                  NA_pid=`cut -f1 $NA_HOME/logs/nodeagent/nodeagent.pid`
                  kill -9 ${NA_pid} > /dev/null 2>&1
                  logger -i -p info -t $0 "NodeAgent (pid ${NA_pid}) killed."
                fi
                ;;
        status)
                NA_UP=`netstat -lnt | grep :${NA_TCPP}`
                if [ "${NA_UP}" != "" ];
                then
                  RC=$ONLINE
                else
                  RC=$OFFLINE
                fi
                ;;
        *)
                logger -i -p info -t $0 "Error: Incorrect parameter
>${Action}<"
                RC=${UNKNOWN}
                ;;
esac
exit ${RC}
```

Example 10-11 wasctrl-as

```
#!/bin/ksh
############################################################
#
# was applicationServer automation control script
#
# Input:
#       $1      action (<start|stop|status>)
#       $2      na_home, the nodeagent's home directory
#               USER_INSTALL_ROOT
#       $3      as_tcpp, the applicationServer's SOAP_CONNECTOR_ADDRESS
#       $4      server_name, name of the applicationServer
#
############################################################
#
# init section
#

UNKNOWN=0
ONLINE=1
OFFLINE=2
```

```
Action=${1:-status}
NA_HOME=$2
AS_TCPP=$3
SERVER_NAME=$4

export
PATH=$PATH:/bin:/usr/bin:/sbin:/usr/sbin:/usr/sbin/rsct/bin:${NA_HOME}/bin

case ${Action} in
        start)
                startServer.sh ${SERVER_NAME} -timeout 300
                RC=$?
                logger -i -p info -t $0 "ApplicationServer start rc: ${RC}"
                ;;
        stop)
                stopServer.sh ${SERVER_NAME} -timeout 180
                RC=$?
                if [ $RC -eq 0 ]; then
                  logger -i -p info -t $0 "ApplicationServer stop rc: ${RC}"
                elif [ $RC -eq 246 ]; then
                  logger -i -p info -t $0 "ApplicationServer stop rc: ${RC}
(applicationServer did not run)."
                else
                  logger -i -p info -t $0 "ApplicationServer stop rc: ${RC}"
                  AS_pid=`cut -f1
$NA_HOME/logs/${SERVER_NAME}/${SERVER_NAME}.pid`
                  kill -9 ${AS_pid} > /dev/null 2>&1
                  logger -i -p info -t $0 "ApplicationServer (pid ${AS_pid})
killed."
                fi
                ;;
        status)
                AS_UP=`netstat -lnt | grep :${AS_TCPP}`
                if [ "${AS_UP}" != "" ];
                then
                  RC=$ONLINE
                else
                  RC=$OFFLINE
                fi
                ;;
        *)
                logger -i -p info -t $0 "Error: Incorrect parameter
>${Action}<"
                RC=${UNKNOWN}
                ;;
esac
exit ${RC}
```

10.4.3 Testing Node Agent and application server failover

When all resources are set up and running on the systems, we recommend you test the setup to see that everything is working as expected.

Using Tivoli System Automation

As mentioned, from now on the Node Agent and application server can only be stopped or started using Tivoli System Automation. The commands that Tivoli System Automation uses to start and stop the Node Agent and application server are shown in Example 10-12.

Example 10-12 Starting and stopping managed Node Agent and application server

```
# command to start the Node Agent
startrsrc -s "Name='SA-was-na'" IBM.Application

# command to stop the Node Agent
stoprsrc -s "Name='SA-was-na'" IBM.Application

# command to stop the application server
stoprsrc -s "Name='SA-was-as'" IBM.Application

# command to start the application server
startrsrc -s "Name='SA-was-as'" IBM.Application
```

You can test failover by first bringing the Node Agent and application server resource group online using the **chrg -o online SA-was-rg** command. This causes Tivoli System Automation to run the start scripts (shown in Example 10-10 on page 386 and Example 10-11 on page 387) for all the resources (and their equivalencies) that were added in Example 10-6 on page 384.

Then, tell Tivoli System Automation to remove the node where the resource group (and subsequently resources) is currently running from the allowed list. Tivoli System Automation keeps a list of nodes that are to be excluded from running a resource group. If the Node Agent and application server is running on thost1, then the following command excludes thost1 from running the resource group. You see the Node Agent and application server shutdown on thost1 and then started on thost2:

```
samctrl -u a thost1
```

This adds thost1 to the excluded node list for the domain, which triggers a orderly failover to thost2. When the Node Agent and application server are running on thost2 then you can undo this command using the following:

```
samctrl -u d thost1
```

This command removes the node thost1 from the disabled node list and makes thost1 available again to running the Node Agent and application server. No failback will occur when the Tivoli System Automation node is added back to the enabled node list. You can fail the Node Agent and application server back to thost1 by first excluding thost2 using **samctrl -u a thost2** and then, when the Node Agent and application server are running on thost1 again, undo the exclude command by adding the node back to the enabled list using **samctrl -u d thost2**.

Testing manually

To test failover manually:

1. Shut down the system where the Node Agent and application server are running.
2. Kill the Node Agent or application server process using the **kill -9** command (UNIX or Linux). The process ids are denoted in the .pid file located in the respective Node Agent and application server logs directories.

10.4.4 Example: Monitoring and restarting two nodes

This example is less of a failover scenario and more of a monitoring and restarting scenario. The Node Agents and application servers will be restarted on their respective node in case of a process failure but will not failover to another node. We describe how to configure IBM Tivoli System Automation to protect the two nodes.

Scenario description

- A WebSphere cluster spans two application servers, each one on a separate node hosting also the Node Agent.
- Both nodes are production nodes in the peer domain.
- The goal is an automatic restart of the Node Agent or application server in case of their failure.
- Tivoli System Automation has one resource group containing four group members. The four group members consist of two Node Agents and two application servers. Each resource group member has a Tivoli System Automation `IBM.Application` definition mapping the start, stop, and monitor scripts for the WebSphere resources.

Resources

- Two Node Agents (one on each peer node).
- Two application servers (one on each node).

The resources are of type fixed.

Network equivalencies

None.

Resource groups

The Node Agent and the application server are members of the same group. There is one group for each peer node defined.

Relationships between resources

The application server depends on the Node Agent running on the same peer node for server start.

Figure 10-3 illustrates the resources and dependencies for the Node Agent and application server monitoring and restarting scenario.

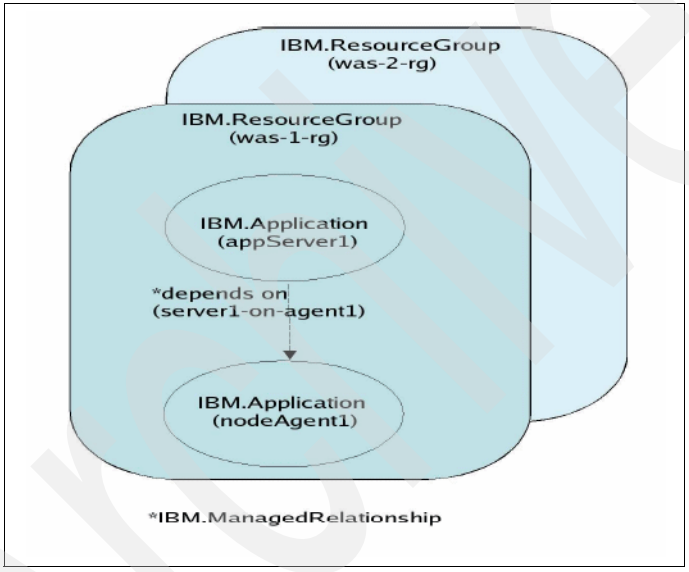


Figure 10-3 Node Agent and application monitoring scenario

Configuring Tivoli System Automation for the monitoring and restarting example

In Example 10-13, notice the resource groups being tied together so that when or if the Node Agent is stopped, the application server on that same node is also stopped.

Example 10-13 Commands to set up sample environment with two nodes

```
# make resource groups
mkrg SA-was-na1-rg

# make resources
mkrsrc -f SA-was-na1.def IBM.Application
mkrsrc -f SA-was-na2.def IBM.Application
mkrsrc -f SA-was-as1.def IBM.Application
mkrsrc -f SA-was-as2.def IBM.Application

# add resources to resource group
addrgmbr -m T -g SA-was-na1-rg IBM.Application:SA-was-na1
addrgmbr -m T -g SA-was-na2-rg IBM.Application:SA-was-na2
addrgmbr -m T -g SA-was-as1-rg IBM.Application:SA-was-as1
addrgmbr -m T -g SA-was-as2-rg IBM.Application:SA-was-as2

# specify dependencies and equivalencies
# No dependencies or equivalencies needed for this scenario
```

Example 10-14 SA-was-na1.def

```
PersistentResourceAttributes::
Name=SA-was-na1
ResourceType=0
StartCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-na start
/opt/IBM/WebSphere/AppServer/profiles/node1 8878
StopCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-na stop
/opt/IBM/WebSphere/AppServer/profiles/node1 8878
MonitorCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-na status
/opt/IBM/WebSphere/AppServer/profiles/node1 8878
StartCommandTimeout=60
StopCommandTimeout=60
MonitorCommandTimeout=9
MonitorCommandPeriod=30
ProtectionMode=1
RunCommandsSync=0
NodeNameList={'thost1'}
UserName=root
```

Example 10-15 SA-was-na2.def

```
PersistentResourceAttributes::
Name=SA-was-na2
ResourceType=0
StartCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-na start
/opt/IBM/WebSphere/AppServer/profiles/node2 8878
StopCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-na stop
/opt/IBM/WebSphere/AppServer/profiles/node2 8878
MonitorCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-na status
/opt/IBM/WebSphere/AppServer/profiles/node2 8878
StartCommandTimeout=60
StopCommandTimeout=60
MonitorCommandTimeout=9
MonitorCommandPeriod=30
ProtectionMode=1
RunCommandsSync=0
NodeNameList={'thost2'}
UserName=root
```

Example 10-16 SA-was-as1.def

```
PersistentResourceAttributes::
Name=SA-was-as1
ResourceType=0
StartCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-as start
/opt/IBM/WebSphere/AppServer/profiles/node1 8879 thost1server
StopCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-as stop
/opt/IBM/WebSphere/AppServer/profiles/node1 8879 thost1server
MonitorCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-as status
/opt/IBM/WebSphere/AppServer/profiles/node1 8879 thost1server
StartCommandTimeout=180
StopCommandTimeout=180
MonitorCommandTimeout=9
MonitorCommandPeriod=30
ProtectionMode=1
RunCommandsSync=0
NodeNameList={'thost1'}
UserName=root
```

Example 10-17 SA-was-as2.def

```
PersistentResourceAttributes::
Name=SA-was-as2
ResourceType=0
StartCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-as start
/opt/IBM/WebSphere/AppServer/profiles/node2 8879 thost2server
StopCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-as stop
/opt/IBM/WebSphere/AppServer/profiles/node2 8879 thost2server
```

```
MonitorCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-as status
/opt/IBM/WebSphere/AppServer/profiles/node2 8879 thost2server
StartCommandTimeout=180
StopCommandTimeout=180
MonitorCommandTimeout=9
MonitorCommandPeriod=30
ProtectionMode=1
RunCommandsSync=0
NodeNameList={'thost2'}
UserName=root
```

Note: For sample wasctrl-na and wasctrl-as scripts, see Example 10-10 on page 386 and Example 10-11 on page 387.

Testing monitoring and restarting example

When all resources are set up and running on the systems we recommend you test the setup to see that everything is working as expected. To test the monitoring and restart scenario, you can kill the Node Agent or application server process using the **kill -9** command (UNIX or Linux). The process IDs are denoted in the .pid file located in the respective Node Agent and application server logs directories.

10.5 Transaction Manager failover with No Operation policy

A general discussion of configuring the Transaction Manager failover with the No Operation policy is provided in “Transaction Manager failover with No Operation policy” on page 313.

In this section, we discuss how to configure IBM Tivoli System Automation to correctly failover the Transaction Manager with the No Operation recovery policy.

Scenario description

- Two production nodes in the peer domain.
- Automatic restart of a Node Agent or application server in case of their failure.
- In addition, in case of an application server failure, this scenario provides failover of the affected Transaction Manager to the other peer node and failback when the failed application server has restarted.
- A shared file system containing the transaction logs that can be accessed by the peer nodes.

Resources

- Two Node Agents (one for each node).
- Two application servers (one for each node).
- Two Transaction Managers (one per node). In order to control the Transaction Managers with Tivoli System Automation, the WebSphere policy must be set to No Operation. In addition, match criteria must be defined for each Transaction Managers instance. For an example look into the sa-was.conf.sample file which is part of the premade policy for WebSphere V6 available at:

  `http://www.ibm.com/software/tivoli/products/sys-auto-linux/downloads.html`

  Note: At the time of writing this redbook, the WebSphere V6 policy and sample script files were not yet available for download. This is planned for the near future. Monitor the Web site for availability.

- Four application server monitors (two for each application server). These monitors report an online operation state as soon as the monitored application server becomes online. And they report offline as long as the monitored application server is not in online state.

The Transaction Manager resources are of type floating, all other resources are of type fixed.

Network equivalencies

None.

Equivalencies of application server monitors

Two equivalencies of application server monitors are defined, one for each Transaction Manager. The SelectFromPolicy attribute of the equivalencies must be set to 27 for this scenario.

Resource groups

- Two for two Node Agent/application server pairs, one for each pair.
- Two for the Transaction Managers, one for each instance.

Relationships between resources

1. The application server depends on the Node Agent.
2. Each Transaction Manager depends on one of the equivalencies of the application server monitors. This establishes an implicit dependency between each of the Transaction Managers (TM) and the application server the TM runs on.

Note: If the two application servers are started simultaneously but take a significantly different amount of time to initialize, both Transaction Managers will be started on the same node. After the “slower” application server has initialized too, its Transaction Manager will be failed back to it.

Figure 10-4 illustrates the resources and dependencies for the Transaction Manager NoOP failover scenario.

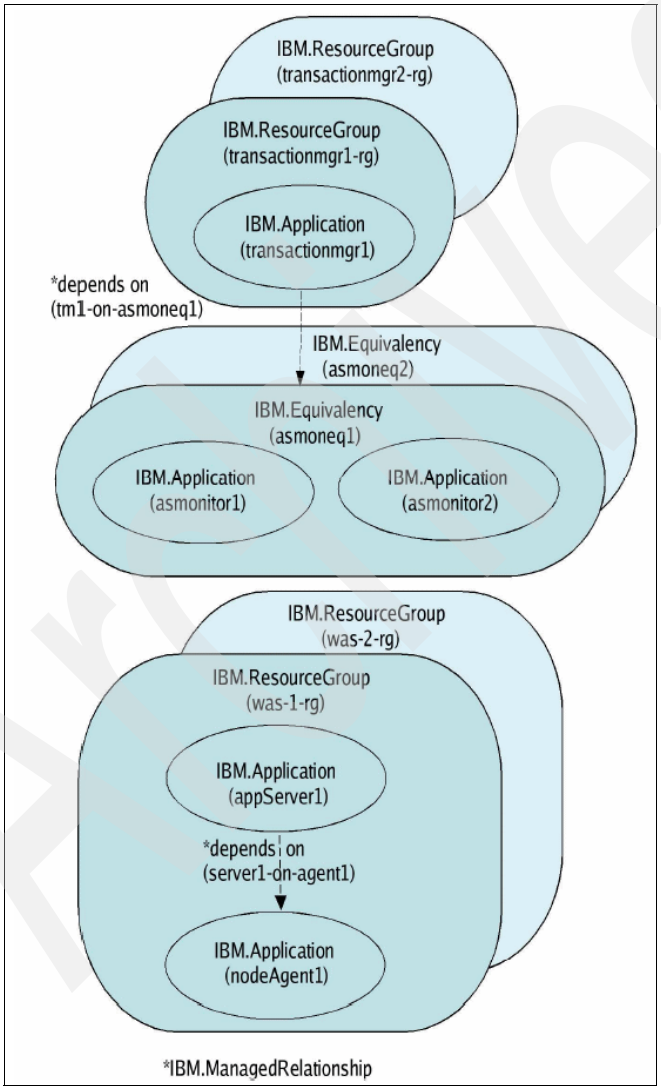


Figure 10-4 Transaction Manager NoOP scenario - resources and dependencies

10.5.1 WebSphere configuration

Follow the instructions in 9.6.3, “Configuring WebSphere for TM No Operation policy” on page 317 to configure WebSphere for this scenario.

10.5.2 Tivoli System Automation configuration

Example 10-18 lists the configuration commands needed to set up the Transaction Manager with NoOP policy scenario in Tivoli System Automation.

Example 10-18 Commands to set up Transaction Manager NoOP scenario

```
# make resource groups
mkrg SA-was-na1-rg
mkrg SA-was-na2-rg
mkrg SA-was-as1-rg
mkrg SA-was-as2-rg
mkrg SA-was-tm1-rg
mkrg SA-was-tm2-rg

# make resources
mkrsrc -f SA-was-na1.def IBM.Application
mkrsrc -f SA-was-na2.def IBM.Application
mkrsrc -f SA-was-as1.def IBM.Application
mkrsrc -f SA-was-as2.def IBM.Application
mkrsrc -f SA-was-tm1.def IBM.Application
mkrsrc -f SA-was-tm2.def IBM.Application
mkrsrc -f SA-was-ascon11.def IBM.Application
mkrsrc -f SA-was-ascon12.def IBM.Application
mkrsrc -f SA-was-ascon21.def IBM.Application
mkrsrc -f SA-was-ascon22.def IBM.Application

# add resources to resource group
addrgmbr -m T -p A -g SA-was-na1-rg IBM.Application:SA-was-na1:thost1
addrgmbr -m T -p A -g SA-was-na2-rg IBM.Application:SA-was-na2:thost2
addrgmbr -m T -p A -g SA-was-as1-rg IBM.Application:SA-was-as1:thost1
addrgmbr -m T -p A -g SA-was-as2-rg IBM.Application:SA-was-as2:thost2
addrgmbr -m T -p A -g SA-was-tm1-rg IBM.Application:SA-was-tm1
addrgmbr -m T -p A -g SA-was-tm2-rg IBM.Application:SA-was-tm2

# specify dependencies and equivalencies
mkequ SA-was-asconeq1
IBM.Application:SA-was-ascon11:thost1,SA-was-ascon12:thost2
chrsrc -s 'Name=\"SA-was-asconeq1\"' IBM.Equivalency SelectFromPolicy=27
mkequ SA-was-asconeq2 IBM.Application:SA-was-ascon22:thost2,SA-was-ascon21:
chrsrc -s 'Name=\"SA-was-asconeq2\"' IBM.Equivalency SelectFromPolicy=27
mkrel -S IBM.Application:SA-was-as1: -G IBM.Application:SA-was-na1:thost2 -p
DependsOn SA-was-as1:thost1-on-na1:thost1
```

```
mkrel -S IBM.Application:SA-was-as2:thost2 -G IBM.Application:SA-was-na2:thost2
-p DependsOn SA-was-as2:thost2-on-na2:thost2
mkrel -S IBM.Application:SA-was-tm1 -G IBM.Equivalency:SA-was-asconeq1 -p
DependsOn SA-was-tm1-on-asconeq1
mkrel -S IBM.Application:SA-was-tm2 -G IBM.Equivalency:SA-was-asconeq2 -p
DependsOn SA-was-tm2-on-asconeq2
```

A sample start, stop, and monitor script (wasctrl-ascon) for the application server resource groups is shown in Example 10-19.

Example 10-19 wasctrl-ascon

```
#!/bin/ksh
############################################################
#
# was tm automation control, auxiliary script
#
# Input:
#       $1      action (<start|stop|status>)
#       $2      application server's resource group name
#       $3      as_tcpp, the applicationServer's SOAP_CONNECTOR_ADDRESS
#
############################################################
#
# init section
#

UNKNOWN=0
ONLINE=1
OFFLINE=2

Action=${1:-status}
ASERVER_RG=$2
AS_TCPP=$3

export PATH=$PATH:/bin:/usr/bin:/sbin:/usr/sbin:/usr/sbin/rsct/bin

case ${Action} in
        start)
                chrg -o online $ASERVER_RG
                RC=$?
                logger -i -p info -t $0 "Resource group $ASERVER_RG start rc:
${RC}"
                ;;
        stop)
                chrg -o offline $ASERVER_RG
                RC=$?
                if [ $RC -eq 0 ]; then
```

```
                  logger -i -p info -t $0 "Resource group $ASERVER_RG stop rc:
${RC}"
                else
                  logger -i -p info -t $0 "Resource group $ASERVER_RG stop
failed with rc: ${RC}"
                fi
                ;;
        status)
                opstate=`lsrg -g ${ASERVER_RG} OpState |awk '/OpState/ {print
$3}'`
                if [ $opstate == "2" ]; then
                  RC=$OFFLINE;
                else
                  AS_UP=`netstat -lnt | grep :${AS_TCPP}`
                  if [ $opstate == "1" -a "${AS_UP}" != "" ]; then
                    RC=$ONLINE;
                  else
                    RC=$OFFLINE;
                  fi
                fi
                ;;
        *)
                logger -i -p info -t $0 "Error: Incorrect parameter
>${Action}<"
                RC=${UNKNOWN}
                ;;
esac

exit ${RC}
```

The examples that follow show the definition files for the commands listed previously.

Note: For sample wasctrl-na and wasctrl-as scripts, see Example 10-10 on page 386 and Example 10-11 on page 387.

Example 10-20 SA-was-na1.def

```
PersistentResourceAttributes::
Name=SA-was-na1
ResourceType=0
StartCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-na start
/opt/IBM/WebSphere/AppServer/profiles/node1 8878
StopCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-na stop
/opt/IBM/WebSphere/AppServer/profiles/node1 8878
MonitorCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-na status
/opt/IBM/WebSphere/AppServer/profiles/node1 8878
```

```
StartCommandTimeout=60
StopCommandTimeout=60
MonitorCommandTimeout=19
MonitorCommandPeriod=30
ProtectionMode=1
RunCommandsSync=0
NodeNameList={'thost1'}
UserName=root
```

Example 10-21 SA-was-na2.def

```
PersistentResourceAttributes::
Name=SA-was-na2
ResourceType=0
StartCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-na start
/opt/IBM/WebSphere/AppServer/profiles/node2 8878
StopCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-na stop
/opt/IBM/WebSphere/AppServer/profiles/node2 8878
MonitorCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-na status
/opt/IBM/WebSphere/AppServer/profiles/node2 8878
StartCommandTimeout=60
StopCommandTimeout=60
MonitorCommandTimeout=19
MonitorCommandPeriod=30
ProtectionMode=1
RunCommandsSync=0
NodeNameList={'thost2'}
UserName=root
```

Example 10-22 SA-was-as1.def

```
PersistentResourceAttributes::
Name=SA-was-as1
ResourceType=0
StartCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-as start
/opt/IBM/WebSphere/AppServer/profiles/node1 8879 thost1server
StopCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-as stop
/opt/IBM/WebSphere/AppServer/profiles/node1 8879 thost1server
MonitorCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-as status
/opt/IBM/WebSphere/AppServer/profiles/node1 8879 thost1server
StartCommandTimeout=180
StopCommandTimeout=180
MonitorCommandTimeout=19
MonitorCommandPeriod=30
ProtectionMode=1
RunCommandsSync=0
NodeNameList={'thost1'}
UserName=root
```

Example 10-23 SA-was-as2.def

```
PersistentResourceAttributes::
Name=SA-was-as2
ResourceType=0
StartCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-as start
/opt/IBM/WebSphere/AppServer/profiles/node2 8879 thost2server
StopCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-as stop
/opt/IBM/WebSphere/AppServer/profiles/node2 8879 thost2server
MonitorCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-as status
/opt/IBM/WebSphere/AppServer/profiles/node2 8879 thost2server
StartCommandTimeout=180
StopCommandTimeout=180
MonitorCommandTimeout=19
MonitorCommandPeriod=30
ProtectionMode=1
RunCommandsSync=0
NodeNameList={'thost2'}
UserName=root
```

Note: The wasctrl-tm script that is used in the next two examples is explained in great detail in 9.6, “Transaction Manager failover with No Operation policy” on page 313, specifically in “Scripts to start, stop, and monitor WebSphere resources” on page 331.

Example 10-24 SA-was-tm1.def

```
PersistentResourceAttributes::
Name=SA-was-tm1
ResourceType=1
StartCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-tm start thost1
/opt/IBM/WebSphere/AppServer/profiles/node1 8879 TradeCluster
StopCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-tm stop thost1
/opt/IBM/WebSphere/AppServer/profiles/node1 8879 TradeCluster
MonitorCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-tm status thost1
/opt/IBM/WebSphere/AppServer/profiles/node1 8879 TradeCluster
StartCommandTimeout=60
StopCommandTimeout=60
MonitorCommandTimeout=9
MonitorCommandPeriod=30
ProtectionMode=1
RunCommandsSync=0
NodeNameList={'thost1','thost2'}
UserName=root
```

Example 10-25 SA-was-tm2.def

```
PersistentResourceAttributes::
Name=SA-was-tm2
ResourceType=1
StartCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-tm start thost2
/opt/IBM/WebSphere/AppServer/profiles/node2 8879 TradeCluster
StopCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-tm stop thost2
/opt/IBM/WebSphere/AppServer/profiles/node2 8879 TradeCluster
MonitorCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-tm status thost2
/opt/IBM/WebSphere/AppServer/profiles/node2 8879 TradeCluster
StartCommandTimeout=60
StopCommandTimeout=60
MonitorCommandTimeout=9
MonitorCommandPeriod=30
ProtectionMode=1
RunCommandsSync=0
NodeNameList={'thost1','thost2'}
UserName=root
```

Note: The following definition files use the wasctrl-ascon script shown in Example 10-19 on page 398.

Example 10-26 SA-was-ascon11.def

```
PersistentResourceAttributes::
Name=SA-was-ascon11
ResourceType=0
StartCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-ascon start SA-was-as1-rg
8879
StopCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-ascon stop SA-was-as1-rg 8879
MonitorCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-ascon status SA-was-as1-rg
8879
StartCommandTimeout=180
StopCommandTimeout=180
MonitorCommandTimeout=19
MonitorCommandPeriod=30
ProtectionMode=1
RunCommandsSync=0
NodeNameList={'thost1'}
UserName=root
```

Example 10-27 SA-was-ascon12.def

```
PersistentResourceAttributes::
Name=SA-was-ascon12
ResourceType=0
StartCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-ascon start SA-was-as2-rg
8879
StopCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-ascon stop SA-was-as2-rg 8879
MonitorCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-ascon status SA-was-as2-rg
8879
StartCommandTimeout=180
StopCommandTimeout=180
MonitorCommandTimeout=19
MonitorCommandPeriod=30
ProtectionMode=1
RunCommandsSync=0
NodeNameList={'thost2'}
UserName=root
```

Example 10-28 SA-was-ascon21.def

```
PersistentResourceAttributes::
Name=SA-was-ascon21
ResourceType=0
StartCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-ascon start SA-was-as1-rg
8879
StopCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-ascon stop SA-was-as1-rg 8879
MonitorCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-ascon status SA-was-as1-rg
8879
StartCommandTimeout=180
StopCommandTimeout=180
MonitorCommandTimeout=19
MonitorCommandPeriod=30
ProtectionMode=1
RunCommandsSync=0
NodeNameList={'thost1'}
UserName=root
```

Example 10-29 SA-was-ascon22.def

```
PersistentResourceAttributes::
Name=SA-was-ascon22
ResourceType=0
StartCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-ascon start SA-was-as2-rg
8879
StopCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-ascon stop SA-was-as2-rg 8879
MonitorCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-ascon status SA-was-as2-rg
8879
StartCommandTimeout=180
StopCommandTimeout=180
MonitorCommandTimeout=19
MonitorCommandPeriod=30
ProtectionMode=1
RunCommandsSync=0
NodeNameList={'thost2'}
UserName=root
```

10.5.3 Testing Transaction Manager with NoOP policy failover

When all resources are set up and running on the systems we recommend you test the setup to see that everything is working as expected. Possible test scenarios are:

1. Shut down the system where the application server is running.
2. Kill the application server process using the **kill -9** command (UNIX or Linux). The process ID is denoted in the .pid file located in the application server logs directory.

Refer to 11.5.3, “Testing Transaction Manager with NoOP policy failover” on page 441 in the HACMP chapter for instructions on how to verify whether your Transaction Manager failed over as expected. Alternatively, you can look at the messages in the SystemOut.log file of the failed-over server where you will find messages indicating the start.

10.6 Default messaging provider with No Operation policy

A general discussion of configuring the default messaging provider failover with the No Operation policy is provided in “Default messaging provider failover with No Operation policy” on page 347.

In this section, we discuss how to configure IBM Tivoli System Automation to correctly failover the messaging engines with the No Operation recovery policy.

Scenario description

- Two production nodes in the peer domain.
- Automatic restart of a Node Agent or application server in case of their failure.
- In addition, in case of an application server failure, failover of the affected messaging engine to the other peer node.

Resources

- Two Node Agents (one for each node).
- Two application servers (one for each node).
- Two messaging engines (one per node). In order to control the messaging engine with Tivoli System Automation the WebSphere policy must be set to No Operation. In addition, match criteria must be defined for each messaging engine instance. For an example look into the sa-was.conf.sample file which is part of the premade policy for WebSphere V6 available at:

  `http://www.ibm.com/software/tivoli/products/sys-auto-linux/downloads.html`

  Note: At the time of writing this redbook, the WebSphere V6 policy and sample script files were not yet available for download. This is planned for the near future. Monitor the Web site for availability.

3. Four application server monitors (two for each application server). These monitors report an online operation state as soon as the monitored application server becomes online. And they report offline as long as the monitored application server is not in online state.

The messaging engine resources are of type floating, all other resources are of type fixed.

Network equivalencies

None.

Equivalencies of application server monitors

Two equivalencies of application server monitors are defined, one for each messaging engine. The SelectFromPolicy attribute of the equivalencies must be set to 25 for this scenario.

Resource Groups

- Two for two Node Agent/application server pairs, one for each pair.
- Two for the messaging engines, one for each instance.

Relationships between resources

- The application server depends on the Node Agent.
- Each messaging engine depends on one of the equivalencies of the application server monitors. This establishes an implicit dependency between each of the messaging engines and the application server the messaging engine runs on.

Note: Under certain circumstances, both messaging engines might be started on the same peer node (see “Relationships between resources” on page 395).

Figure 10-5 on page 407 illustrates the resources and dependencies for the messaging engine NoOP failover scenario.

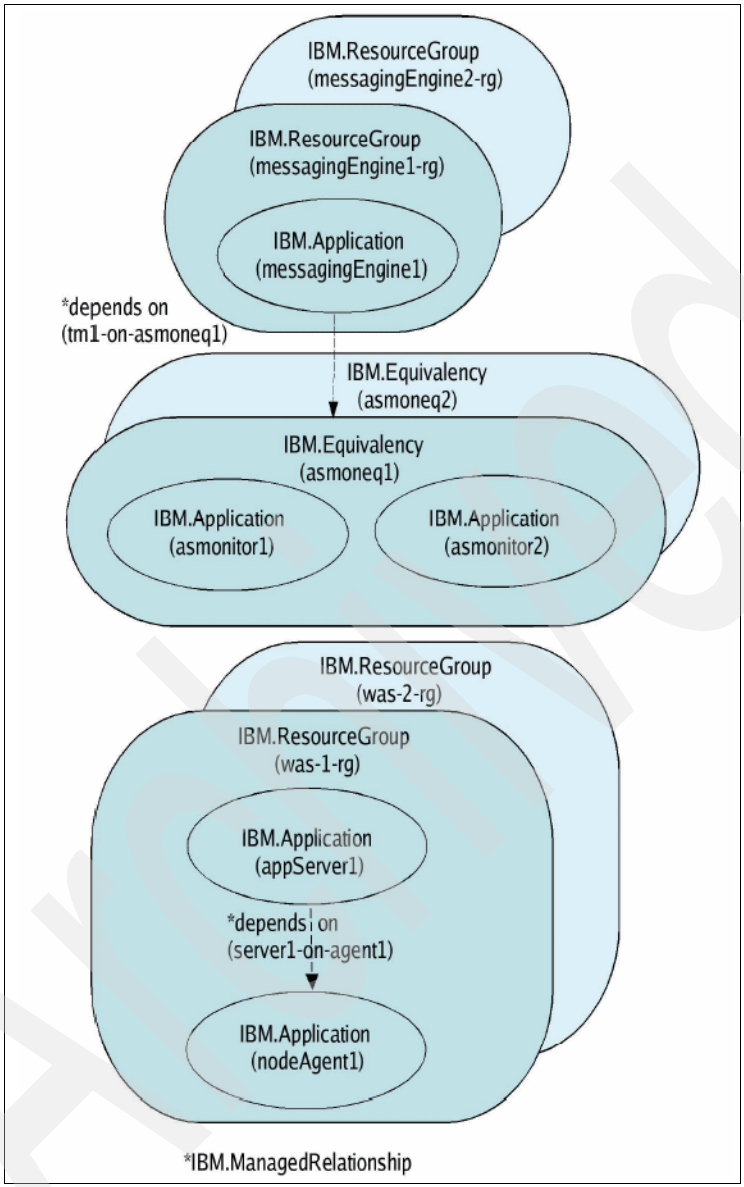


Figure 10-5 Messaging engine NoOP scenario - resources and dependencies

10.6.1 WebSphere configuration

Follow the instructions in 9.7.3, “Configuring WebSphere for default messaging provider No Operation policy” on page 349 to configure WebSphere for this scenario.

10.6.2 Tivoli System Automation configuration

Example 10-30 lists the configuration commands needed to set up the messaging engine with NoOP policy scenario in Tivoli System Automation.

Example 10-30 Commands to set up messaging engine NoOP scenario

```
# make resource groups
mkrg SA-was-1-rg
mkrg SA-was-2-rg
mkrg SA-was-me1-rg
mkrg SA-was-me2-rg

# make resources
mkrsrc -f SA-was-na1.def IBM.Application
mkrsrc -f SA-was-na2.def IBM.Application
mkrsrc -f SA-was-as1.def IBM.Application
mkrsrc -f SA-was-as2.def IBM.Application
mkrsrc -f SA-was-me1.def IBM.Application
mkrsrc -f SA-was-me2.def IBM.Application
mkrsrc -f SA-was-ascon11.def IBM.Application
mkrsrc -f SA-was-ascon12.def IBM.Application
mkrsrc -f SA-was-ascon21.def IBM.Application
mkrsrc -f SA-was-ascon22.def IBM.Application

# specify dependencies and equivalencies
addrgmbr -m T -p A -g SA-was-1-rg IBM.Application:SA-was-na1:thost1
addrgmbr -m T -p A -g SA-was-2-rg IBM.Application:SA-was-na2:thost2
addrgmbr -m T -p A -g SA-was-1-rg IBM.Application:SA-was-as1:thost1
addrgmbr -m T -p A -g SA-was-2-rg IBM.Application:SA-was-as2:thost2
addrgmbr -m T -p A -g SA-was-me1-rg IBM.Application:SA-was-me1
addrgmbr -m T -p A -g SA-was-me2-rg IBM.Application:SA-was-me2

# specify dependencies and equivalencies
mkequ SA-was-asconeq1
IBM.Application:SA-was-ascon11:thost1,SA-was-ascon12:thost2
chrsrc -s  'Name="SA-was-asconeq1"' IBM.Equivalency SelectFromPolicy=25
mkequ SA-was-asconeq2
IBM.Application:SA-was-ascon22:thost2,SA-was-ascon21:thost1
chrsrc -s  'Name="SA-was-asconeq2"' IBM.Equivalency SelectFromPolicy=25
mkrel -S IBM.Application:SA-was-as1:thost1 -G IBM.Application:SA-was-na1:thost1
-p DependsOn  SA-was-as1:thost1-on-na1:thost1
```

```
mkrel -S IBM.Application:SA-was-as2:thost2 -G IBM.Application:SA-was-na2:thost2
-p DependsOn  SA-was-as2:thost2-on-na2:thost2
mkrel -S IBM.Application:SA-was-me1 -G IBM.Equivalency:SA-was-asconeq1 -p
DependsOn  SA-was-me1-on-asconeq1
mkrel -S IBM.Application:SA-was-me2 -G IBM.Equivalency:SA-was-asconeq2 -p
DependsOn  SA-was-me2-on-asconeq2
```

The following examples show the definition files for the commands listed previously.

Note: For sample wasctrl-na and wasctrl-as scripts see Example 10-10 on page 386 and Example 10-11 on page 387.

Example 10-31 SA-was-na1.def

```
PersistentResourceAttributes::
Name=SA-was-na1
ResourceType=0
StartCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-na start
/opt/IBM/WebSphere/AppServer/profiles/node1 8878
StopCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-na stop
/opt/IBM/WebSphere/AppServer/profiles/node1 8878
MonitorCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-na status
/opt/IBM/WebSphere/AppServer/profiles/node1 8878
StartCommandTimeout=60
StopCommandTimeout=60
MonitorCommandTimeout=19
MonitorCommandPeriod=30
ProtectionMode=1
RunCommandsSync=0
NodeNameList={'thost1'}
UserName=root
```

Example 10-32 SA-was-na2.def

```
PersistentResourceAttributes::
Name=SA-was-na2
ResourceType=0
StartCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-na start
/opt/IBM/WebSphere/AppServer/profiles/node2 8878
StopCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-na stop
/opt/IBM/WebSphere/AppServer/profiles/node2 8878
MonitorCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-na status
/opt/IBM/WebSphere/AppServer/profiles/node2 8878
StartCommandTimeout=60
StopCommandTimeout=60
MonitorCommandTimeout=19
MonitorCommandPeriod=30
ProtectionMode=1
RunCommandsSync=0
NodeNameList={'thost2'}
UserName=root
```

Example 10-33 SA-was-as1.def

```
PersistentResourceAttributes::
Name=SA-was-as1
ResourceType=0
StartCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-as start
/opt/IBM/WebSphere/AppServer/profiles/node1 8879 thost1server
StopCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-as stop
/opt/IBM/WebSphere/AppServer/profiles/node1 8879 thost1server
MonitorCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-as status
/opt/IBM/WebSphere/AppServer/profiles/node1 8879 thost1server
StartCommandTimeout=180
StopCommandTimeout=180
MonitorCommandTimeout=19
MonitorCommandPeriod=30
ProtectionMode=1
RunCommandsSync=0
NodeNameList={'thost1'}
UserName=root
```

Example 10-34 SA-was-as2.def

```
PersistentResourceAttributes::
Name=SA-was-as2
ResourceType=0
StartCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-as start
/opt/IBM/WebSphere/AppServer/profiles/node2 8879 thost2server
StopCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-as stop
/opt/IBM/WebSphere/AppServer/profiles/node2 8879 thost2server
MonitorCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-as status
/opt/IBM/WebSphere/AppServer/profiles/node2 8879 thost2server
StartCommandTimeout=180
StopCommandTimeout=180
MonitorCommandTimeout=19
MonitorCommandPeriod=30
ProtectionMode=1
RunCommandsSync=0
NodeNameList={'thost2'}
UserName=root
```

Note: The wasctrl-me script used in Example 10-35 and Example 10-36 is explained in great detail in 9.7, “Default messaging provider failover with No Operation policy” on page 347, specifically in “Scripts to start, stop, and monitor WebSphere resources” on page 357.

Example 10-35 SA-was-me1.def

```
PersistentResourceAttributes::
Name=SA-was-me1
ResourceType=1
StartCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-me start
/opt/IBM/WebSphere/AppServer/profiles/node1
/opt/IBM/WebSphere/AppServer/profiles/node2 /usr/sbin/rsct/sapolicies/was 8879
9080
WSAF_SIB_BUS=TradeCluster,WSAF_SIB_MESSAGING_ENGINE=TradeCluster.000-TradeClust
er,IBM_hc=TradeCluster,type=WSAF_SIB
StopCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-me stop
/opt/IBM/WebSphere/AppServer/profiles/node1
/opt/IBM/WebSphere/AppServer/profiles/node2 /usr/sbin/rsct/sapolicies/was 8879
9080
WSAF_SIB_BUS=TradeCluster,WSAF_SIB_MESSAGING_ENGINE=TradeCluster.000-TradeClust
er,IBM_hc=TradeCluster,type=WSAF_SIB
MonitorCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-me status
/opt/IBM/WebSphere/AppServer/profiles/node1
/opt/IBM/WebSphere/AppServer/profiles/node2 /usr/sbin/rsct/sapolicies/was 8879
9080
WSAF_SIB_BUS=TradeCluster,WSAF_SIB_MESSAGING_ENGINE=TradeCluster.000-TradeClust
er,IBM_hc=TradeCluster,type=WSAF_SIB
```

```
StartCommandTimeout=60
StopCommandTimeout=60
MonitorCommandTimeout=9
MonitorCommandPeriod=30
ProtectionMode=1
RunCommandsSync=0
NodeNameList={'thost1','thost2'}
UserName=root
```

Example 10-36 SA-was-me2.def

```
PersistentResourceAttributes::
Name=SA-was-me2
ResourceType=1
StartCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-me start
/opt/IBM/WebSphere/AppServer/profiles/node2
/opt/IBM/WebSphere/AppServer/profiles/node1 /usr/sbin/rsct/sapolicies/was 8879
9080
WSAF_SIB_BUS=TradeCluster,WSAF_SIB_MESSAGING_ENGINE=TradeCluster.000-TradeClust
er,IBM_hc=TradeCluster,type=WSAF_SIB
StopCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-me stop
/opt/IBM/WebSphere/AppServer/profiles/node2
/opt/IBM/WebSphere/AppServer/profiles/node1 /usr/sbin/rsct/sapolicies/was 8879
9080
WSAF_SIB_BUS=TradeCluster,WSAF_SIB_MESSAGING_ENGINE=TradeCluster.000-TradeClust
er,IBM_hc=TradeCluster,type=WSAF_SIB
MonitorCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-me status
/opt/IBM/WebSphere/AppServer/profiles/node2
/opt/IBM/WebSphere/AppServer/profiles/node1 /usr/sbin/rsct/sapolicies/was 8879
9080
WSAF_SIB_BUS=TradeCluster,WSAF_SIB_MESSAGING_ENGINE=TradeCluster.000-TradeClust
er,IBM_hc=TradeCluster,type=WSAF_SIB
StartCommandTimeout=60
StopCommandTimeout=60
MonitorCommandTimeout=9
MonitorCommandPeriod=30
ProtectionMode=1
RunCommandsSync=0
NodeNameList={'thost2','thost1'}
UserName=root
```

Note: The following definition files use the wasctrl-ascon script that is shown in Example 10-19 on page 398.

Example 10-37 SA-was-ascon11.def

```
PersistentResourceAttributes::
Name=SA-was-ascon11
ResourceType=0
StartCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-ascon start SA-was-1-rg 8879
StopCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-ascon stop SA-was-1-rg 8879
MonitorCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-ascon status SA-was-1-rg
8879
StartCommandTimeout=180
StopCommandTimeout=180
MonitorCommandTimeout=19
MonitorCommandPeriod=30
ProtectionMode=1
RunCommandsSync=0
NodeNameList={'thost1'}
UserName=root
```

Example 10-38 SA-was-ascon12.def

```
PersistentResourceAttributes::
Name=SA-was-ascon12
ResourceType=0
StartCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-ascon start SA-was-2-rg 8879
StopCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-ascon stop SA-was-2-rg 8879
MonitorCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-ascon status SA-was-2-rg
8879
StartCommandTimeout=180
StopCommandTimeout=180
MonitorCommandTimeout=19
MonitorCommandPeriod=30
ProtectionMode=1
RunCommandsSync=0
NodeNameList={'thost2'}
UserName=root
```

Example 10-39 SA-was-ascon21.def

```
PersistentResourceAttributes::
Name=SA-was-ascon21
ResourceType=0
StartCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-ascon start SA-was-1-rg 8879
StopCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-ascon stop SA-was-1-rg 8879
MonitorCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-ascon status SA-was-1-rg
8879
StartCommandTimeout=180
StopCommandTimeout=180
MonitorCommandTimeout=19
MonitorCommandPeriod=30
ProtectionMode=1
RunCommandsSync=0
NodeNameList={'thost1'}
UserName=root
```

Example 10-40 SA-was-ascon22.def

```
PersistentResourceAttributes::
Name=SA-was-ascon22
ResourceType=0
StartCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-ascon start SA-was-2-rg 8879
StopCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-ascon stop SA-was-2-rg 8879
MonitorCommand=/usr/sbin/rsct/sapolicies/was/wasctrl-ascon status SA-was-2-rg
8879
StartCommandTimeout=180
StopCommandTimeout=180
MonitorCommandTimeout=19
MonitorCommandPeriod=30
ProtectionMode=1
RunCommandsSync=0
NodeNameList={'thost2'}
UserName=root
```

10.6.3 Testing messaging engine with NoOP policy failover

When all the resources are set up and running on the systems we recommend you test the setup to see that everything is working as expected. Possible tests include:

1. Shut down the system where the application server is running.
2. Kill the application server process using the **`kill -9`** command (UNIX or Linux). The process ID is denoted in the .pid file located in the application server logs directory.

When a messaging engine starts in a server, messages are written to the SystemOut.log file. Therefore, monitor this file on the failed-over system for messages similar to the ones shown in Example 10-41.

Example 10-41 Message engine startup messages

```
[8/25/05 11:18:03:743 CDT] 00000045 SibMessage    I
[TradeCluster:TradeCluster.000-TradeCluster] CWSIS1538I: The messaging engine,
ME_UUID=02BF0BF501CE0540, INC_UUID=373ad6a9ee6e8153, is attempting to obtain an
exclusive lock on the data store.
...
[8/25/05 11:18:04:037 CDT] 00000045 SibMessage    I
[TradeCluster:TradeCluster.000-TradeCluster] CWSIS1537I: The messaging engine,
ME_UUID=02BF0BF501CE0540, INC_UUID=373ad6a9ee6e8153, has acquired an exclusive
lock on the data store.
...
[8/25/05 11:18:11:227 CDT] 00000027 SibMessage    I
[TradeCluster:TradeCluster.000-TradeCluster] CWSID0016I: Messaging engine
TradeCluster.000-TradeCluster is in state Started.
```

Alternatively, you can open the Administrative Console and go to **Core groups** → **Core group settings** → ***CoreGroupName*** (most probably DefaultCoreGroup) → **Runtime tab**.

Enter `WSAF_SIB_MESSAGING_ENGINE=`*`Your_ME`* into the Group name properties field and click **Show groups.** (For example, WSAF_SIB_MESSAGING_ENGINE=TradeCluster.000-TradeCluster.) You could also leave the asterisk (*) in the Group name properties field and click **Show groups** to display all High availability groups.

Click the link for your HA group to display the Name of the Server the ME runs on. This panel also shows the status of the ME. See Figure 10-6. Verify this panel before and after the failover.

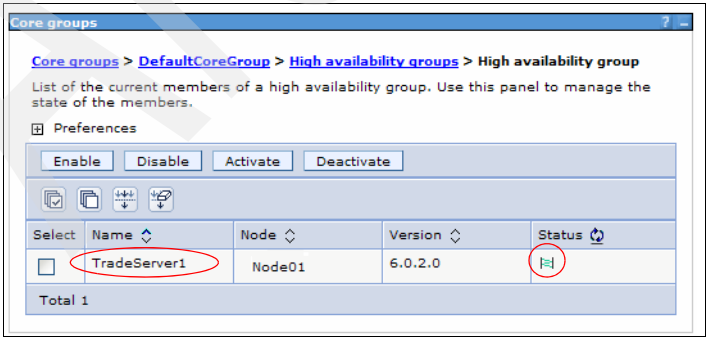


Figure 10-6 ME status and server it runs on

10.7 Reference

- IBM Tivoli System Automation for Multiplatforms Web site

  http://www.ibm.com/software/tivoli/products/sys-auto-linux/

- *IBM Tivoli System Automation for Multiplatforms Guide and Reference*

  http://publib.boulder.ibm.com/tividd/td/ITSAFL/SC33-8210-03/en_US/PDF/halgre11.pdf

- IBM Tivoli System Automation for Multiplatforms, Downloads (papers and scripts)

  http://www.ibm.com/software/tivoli/products/sys-auto-linux/downloads.html

11

WebSphere and IBM HACMP

This chapter provides an introduction into the basics of clustering IBM WebSphere Application Server V6 using IBM High Availability Clustered Multi-Processing (HACMP). We describe how to set up IBM WebSphere Application Server Network Deployment V6 to leverage the benefits of IBM HACMP.

11.1 Introduction to IBM HACMP

In general, high availability is achieved by making systems redundant. The more system redundancy, the higher the level of availability that can be achieved. IBM HACMP for AIX provides a highly available computing environment by adding software and redundant hardware components. It automatically switches applications and data from one system to another in an HACMP cluster after a hardware or software failure. WebSphere Application Server, coupled with IBM HACMP for AIX, delivers a proven and reliable software portfolio for mission-critical On Demand Business applications.

This chapter provides a set of examples of how to set up IBM WebSphere Application Server Network Deployment V6 in an HACMP environment and how to configure WebSphere Application Server to failover successfully. It is not our intention to document the complete HACMP setup, including planning, designing, customization, installation, and configuration. For more detailed information about HACMP setup, refer to the HACMP documentation available at:

http://www.ibm.com/servers/eserver/pseries/library/hacmp_docs.html

We conducted a series of tests to demonstrate how several WebSphere Application Server Network Deployment components achieve high availability through the configuration of HACMP. We examined the following WebSphere Application Server components:

- Deployment Manager
- Node Agent and application servers configured to use the default messaging provider and an application with two-phase commit transactions to recover messages and in-doubt transactions.
- Transaction Manager failover using the NoOP policy (see 9.6, “Transaction Manager failover with No Operation policy” on page 313 for a generic description of this scenario).

11.1.1 How HACMP works

Our HACMP configuration was a typical configuration of HACMP’s *cascading resource group*. Resources move through an ordered list of nodes from the highest to lowest priority in the group. In our configuration, all resources in the group are moved from the primary machine to its standby machine when the primary machine fails. During a failure, such as a network or hardware failure, HACMP on the primary machine notifies its peer services on the standby machine through the heartbeat communication. HACMP on the standby machine recognizes the failure event. It takes over the service IP address of the primary machine, mounts the shared file system and starts all registered servers, such as

WebSphere Application Server. For our tests, IBM HACMP/ES 4.5.0.3 was installed on two systems with AIX V5.1 (note that the latest HACMP version is V5.2). All tests described in this chapter were performed using the same HACMP configuration.

We show the HACMP failover flow in Figure 11-1 and Figure 11-2 on page 420. Before the failure (Figure 11-1), the disk array is attached to the primary machine, called Machine1. WebSphere services run on the primary machine. Clients make requests to the primary machine using the virtual network interface called hacmp1.

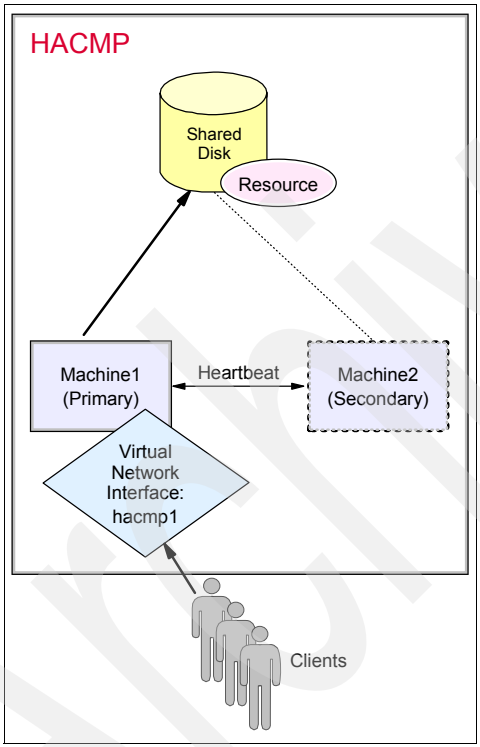


Figure 11-1 Before failover

When a hardware or software failure happens, the HACMP system detects the failure and executes the failover procedure. After the failover, as shown in Figure 11-2, the disk array and the virtual network interface are attached to the secondary machine, Machine2. The WebSphere services now run on the secondary machine.

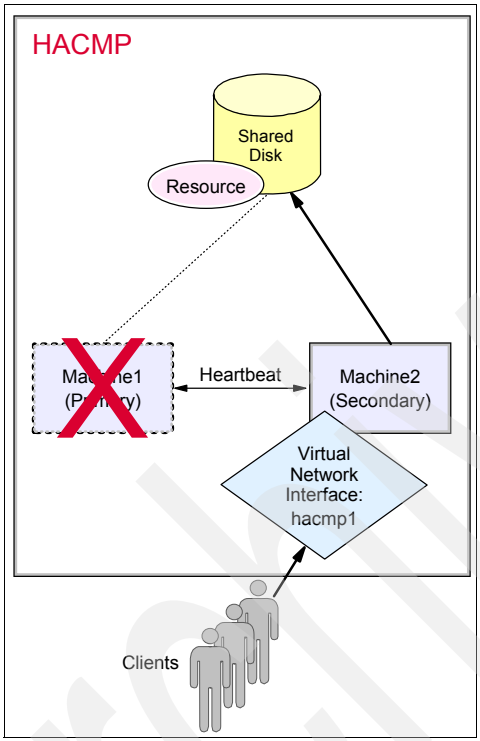


Figure 11-2 After failover

11.1.2 Configuration basics of HACMP

Our HACMP cluster consists of two IBM @server pSeries systems attached to an SSA 7133-D40 disk rack used for sharing the data. The hardware configuration details are as follows:

- Primary machine HHOST1 - 1 CPU, 2GB memory
- Standby machine HHOST2 - 4 CPUs, 4GB memory
- One IP network — Ethernet to external network
- One non-IP network — Serial RS232 for heartbeat between Machine1 and Machine2

The defined HACMP cluster is called hacmpcluster. Its cluster ID is 77. This cluster has two nodes (HHOST1 and HHOST2), one resource group (hacmpgroup), and one HACMP application server (hacmpas). Note that here the term *application server* means the unit of failover in HACMP.

Example 11-1 shows the query result for the HACMP cluster topology, including the names of the nodes, network definitions, and other pertinent information. Note that hacmp1 is the name of the service interface on Machine1. To display the cluster topology, follow these steps:

1. Enter `smitty hacmp` in an operating system command prompt and press Enter.
2. Select **Cluster Configuration**.
3. Select **Cluster Topology**.
4. Select **Show Cluster Topology**.
5. Select **Show Cluster Topology** again.

Example 11-1 The result of Show Cluster Topology

```
[TOP]
Cluster Description of Cluster hacmpcluster
Cluster ID: 77
There were 2 networks defined: haether, haserial
There are 2 nodes in this cluster.
NODE hnode1:
This node has 2 service interface(s):
Service Interface hacmp1:
                IP address:     10.0.2.1
                Hardware Address:
                Network:        haether
                Attribute:      public
                Aliased Address?:       Not Supported
Service Interface hacmp1 has 1 boot interfaces.
                Boot (Alternate Service) Interface 1: hacmp1bt
                IP address:     10.0.2.2
                Network:        haether
                Attribute:      public
Service Interface hacmp1 has 1 standby interfaces.
                Standby Interface 1: hacmp1sb
                IP address:     10.10.0.11
                Network:        heather

Service Interface hacmp1tty:
                IP address:     /dev/tty0
                Hardware Address:
                Network:        haserial
                Attribute:      serial
```

```
                Aliased Address?:       Not Supported
Service Interface hacmp1tty has no boot interfaces.
Service Interface hacmp1tty has no standby interfaces.

NODE hnode2:
This node has 2 service interface(s):
Service Interface hacmp2:
                IP address:     10.0.2.3
                Hardware Address:
                Network:        haether
                Attribute:      public
                Aliased Address?:       Not Supported

Service Interface hacmp2 has 1 boot interfaces.
                Boot (Alternate Service) Interface 1: hacmp2bt
                IP address:     10.0.2.4
                Network:        haether
                Attribute:      public
Service Interface hacmp2 has 1 standby interfaces.
                Standby Interface 1: hacmp2sb
                IP address:     10.10.0.12
                Network:        haether
                Attribute:      public
Service Interface hacmp2tty:
                IP address:     /dev/tty0
                Hardware Address:
                Network:        haserial
                Attribute:      serial
                Aliased Address?:       Not Supported

Service Interface hacmp2tty has no boot interfaces.
Service Interface hacmp2tty has no standby interfaces.

Breakdown of network connections:
Connections to network haether
Node hnode1 is connected to network haether by these interfaces:
                hacmp1bt
                hacmp1
                hacmp1sb
Node hnode2 is connected to network haether by these interfaces:
                hacmp2bt
                hacmp2
                hacmp2sb
Connections to network haserial
Node hnode1 is connected to network haserial by these interfaces:
                hacmp1tty
Node hnode2 is connected to network haserial by these interfaces:
                hacmp2tty
[BOTTOM]
```

11.1.3 Managing resources

The nodes have a shared external disk, with only one node accessing the disk at a time. Both nodes can access a volume group (havg), and a file system (/usr/WebSphere6).

HACMP resource information

Example 11-2 shows the query result for the HACMP resource information of the resource group, including the resource groups and the shared file system. To display this information for your HACMP environment, follow these steps:

1. Enter `smitty hacmp` in an operating system command prompt and press Enter.
2. Select **Cluster Configuration**.
3. Select **Cluster Resources**.
4. Select **Show Cluster Resources**.
5. Select **Show Resource Information by Resource Group**.
6. Select your defined ***Resource_Group_Name***.

Example 11-2 The result of Resource Information query

```
[TOP]
Resource Group Name                          hacmpgroup
Node Relationship                            cascading
Site Relationship                            ignore
Participating Node Name(s)                   hnode1 hnode2
Dynamic Node Priority
Service IP Label hacmp1
Filesystems                                  /usr/WebSphere6
Filesystems Consistency Check                fsck
Filesystems Recovery Method                  sequential
Filesystems/Directories to be exported
Filesystems to be NFS mounted
Filesystems to be NFS mounted
Network For NFS Mount
Volume Groups                                havg
Concurrent Volume Groups
Disks
GMD Replicated Resources
PPRC Replicated Resources
Connections Services
Fast Connect Services
Shared Tape Resources
Application Servers                          hacmpas
Highly Available Communication Links
Primary Workload Manager Class
```

```
Secondary Workload Manager Class
Miscellaneous Data
Automatically Import Volume Groups          false
Inactive Takeover                           false
Cascading Without Fallback                  false
SSA Disk Fencing                            false
Filesystems mounted before IP configured    false
Run Time Parameters:
Node Name                                   hnode1
Debug Level                                 high
Format for hacmp.out                        Standard
Node Name                                   hnode2
Debug Level                                 high
Format for hacmp.out                        Standard
[BOTTOM]
```

HACMP application server information

Example 11-3 shows the query result for the HACMP application servers. To display this information for your environment, do the following:

1. Enter `smitty hacmp` in an operating system command prompt and press Enter.
2. Select **Cluster Configuration**.
3. Select **Cluster Resources**.
4. Select **Define Application Servers**.
5. Select **Change / Show an Application Server**.
6. Select your defined ***Application_Server_Name***.

Example 11-3 The result of querying the application server information

```
Server Name                                 hacmpas
New Server Name                            [hacmpasas]
Start Script                              [/usr/bin/ha.start]
Stop Script                               [/usr/bin/ha.stop]
```

Sample start and stop scripts

Our Start Script is called ha.start, and our Stop Script is called ha.stop, as shown in Example 11-3 on page 424. The content of these scripts varies depending on which WebSphere component is made highly available using the HACMP cluster — the Deployment Manager, the Node Agent and application servers. The following examples show the content of the start and stop scripts that are used for the various scenarios.

Deployment Manager failover scripts

Example 11-4 Sample ha.start script for Deployment Manager failover

```
/usr/WebSphere6/AppServer/profiles/DmgrO1/bin/startManager.sh
```

Example 11-5 Sample ha.stop script for Deployment Manager failover

```
/usr/WebSphere6/AppServer/profiles/DmgrO1/bin/stopManager.sh
```

Node Agent and application server failover scripts

Example 11-6 Sample ha.start script for application server node

```
/usr/WebSphere6/AppServer/profiles/CustomO1/bin/startNode.sh
sleep 60
/usr/WebSphere6/AppServer/profiles/CustomO1/bin/startServer.sh wasmemberO7
/usr/WebSphere6/AppServer/profiles/CustomO1/bin/startServer.sh wasmemberO8
```

Example 11-7 Sample ha.stop script for application server node

```
/usr/WebSphere6/AppServer/profiles/CustomO1/bin/stopServer.sh wasmemberO7
/usr/WebSphere6/AppServer/profiles/CustomO1/bin/stopServer.sh wasmemberO8
/usr/WebSphere6/AppServer/profiles/CustomO1/bin/stopNode.sh
```

The scripts for the Transaction Manager No Operation policy failover are more complicated and call other scripts inside and are thus described in detail in 11.5, “Transaction Manager failover with No Operation policy” on page 436.

HACMP Application Monitor Information

If you need to monitor a specific WebSphere Application Server process or listening network port, you can use the HACMP Application Monitor function. We use a script that monitors the application server’s listening port.

Example 11-8 shows the query result for the HACMP application monitor. To display this information, do the following:

1. Enter `smitty hacmp` in an operating system command prompt and press Enter.
2. Select **Cluster Configuration**.
3. Select **Cluster Resources**.
4. Select **Configure Application Monitoring**.
5. Select **Define Custom Application Monitor**.
6. Select **Change/Show Custom Application Monitor**.
7. Select your defined ***Application_Server_Name***.

Example 11-8 The result of displaying Monitoring Information

```
Application Server Name                          hacmpas
Monitor Method                                   [/usr/bin/ha.monitor]
Monitor Interval                                 [10]
Hung Monitor Signal                              [9]
Stabilization Interval                           [180]
Restart Count                                    [0]
Restart Interval                                 [0]
Action on Application Failure                    [failover]
Notify Method                                    []
Cleanup Method                                   [/usr/bin/ha.stop]
Restart Method                                   [/usr/bin/ha.start]
```

We use the ha.monitor script shown in Example 11-9. The HACMP system executes this script periodically, and if the return code is not zero (0), HACMP executes a failover. In this script , we check the TCP/IP listening port of the Deployment Manager or application server to determine whether the Deployment Manager or application server is running. We use the Deployment Manager's SOAP port and the application server's WebContainer Inbound Chain port respectively.

Example 11-9 Sample ha.monitor script

```
#!/bin/ksh
AS_UP=`netstat -at | grep *.<monitoring_tcpip_portnumber>`
if [ "${AS_UP}" != "" ];
then
   RC=0
else
   RC=1
fi
exit ${RC}
```

11.1.4 Using WebSphere MQ SupportPac for HACMP

Configuration and implementation information about how to make IBM WebSphere MQ highly available using HACMP is provided in the *MQSeries for AIX - Implementing with HACMP Version 2.0* guide, which is available at:

ftp://ftp.software.ibm.com/software/integration/support/supportpacs/individual/mc63.pdf

11.1.5 Using DB2 with HACMP

Configuration and implementation information about how to make IBM DB2 Universal Database™ highly available using HACMP is provided in the *IBM DB2 Universal Database Enterprise Edition for AIX and HACMP/ES* guide, which is available at:

ftp://ftp.software.ibm.com/software/data/pubs/papers/db2ee-aixhacmp.pdf

Also refer to the DB2 Information Center at:

http://publib.boulder.ibm.com/infocenter/db2help/index.jsp

11.2 Planning and preparation

HACMP planning and configuration itself is not related to the WebSphere configuration. Refer to the HACMP planning and installation guides for the different HACMP versions to find information about how to set up your HACMP cluster. These guides are available at:

http://www.ibm.com/servers/eserver/pseries/library/hacmp_docs.html

We assume the following prerequisites are met before starting the WebSphere configuration:

- You have an HACMP cluster configured that consists of at least two systems which share a storage device.
- You have checked the prerequisites needed for IBM WebSphere Application Server Network Deployment V6 at:

  http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html

  Note: At the time of writing this redbook, the officially supported configuration is running the Deployment Manager on HACMP/ES 5.1.

- You have an IP Alias or virtual host name.

- You have determined how you are going to configure WebSphere. Specifically, you have determined whether you are going to separate the binary files from the configuration files. You have two options:
  - You can put the binaries and configuration files on the shared disk.
  - You can put the binaries on a local disk and the configuration files on the shared disk.

11.3 Deployment Manager

We configured the HACMP environment shown in Figure 11-3 for the Deployment Manager failover scenario. The WebSphere product binaries are installed on the shared file system.

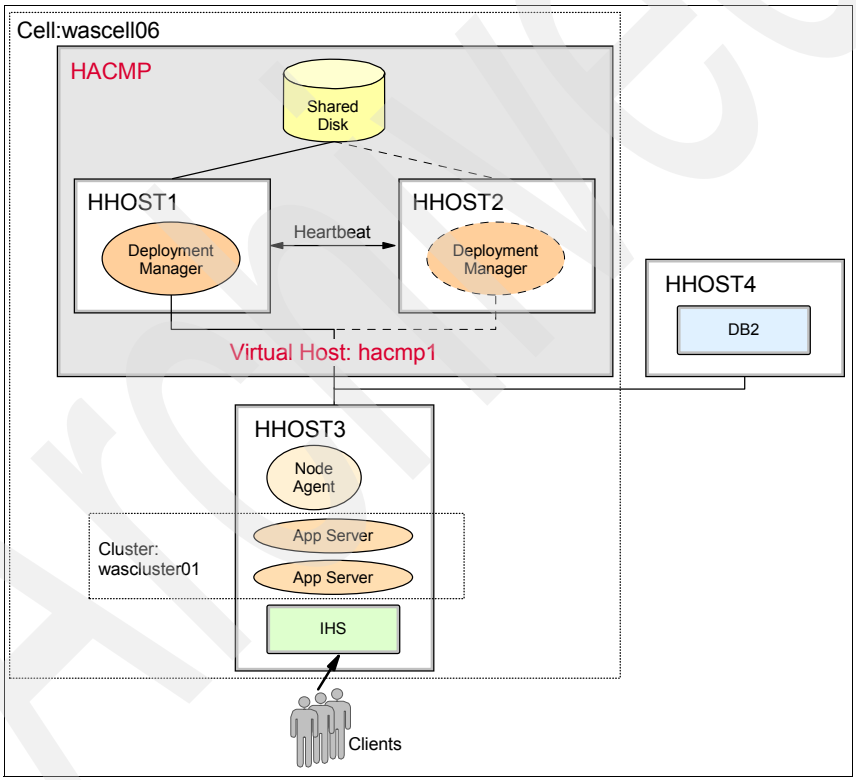


Figure 11-3 HACMP Deployment Manager failover test scenario

11.3.1 Installing the Deployment Manager

To install the Deployment Manager and create the profile, do the following:

1. Start the HACMP services on HHOST1 to mount the file system /usr/WebSphere6 from the shared disk array.
2. Install IBM WebSphere Application Server Network Deployment V6 on HHOST1. The installation path is on the shared file system, in our scenario the path is /usr/WebSphere6/AppServer.
3. If necessary, install any needed WebSphere Refresh packs or Fix packs. We installed WebSphere V6.0.1 for our tests.
4. Create a Deployment Manager profile on HHOST1. In our environment, where the binaries are installed on the shared file system, the profile directory is a subdirectory of the WebSphere <*install_root*>, for example /usr/WebSphere6/AppServer/profiles/Dmgr01.

   When creating the profile, use the virtual host name (in our environment this is hacmp1) as the Host name, you must not use the physical host name or physical IP address. See 9.3.2, “Installing WebSphere Application Server Network Deployment” on page 301 for additional information.
5. Start the Deployment Manager and add nodes to the cell as needed. Use the virtual host name hacmp1 when you specify the Host name or IP address of the Deployment Manager. In our example, we federated node HHOST3 into the cell managed by the Deployment Manager on HHOST1.

11.3.2 Configuring HACMP to run the Deployment Manager

To configure HACMP to run the Deployment Manager, do the following:

1. Add the Deployment Manager stop and start commands to the respective HACMP stop (ha.stop) and start (ha.start) scripts on HHOST1 and HHOST2. Our Deployment Manager stop and start scripts are shown in Example 11-10 and Example 11-11.

Example 11-10 Script to stop the Deployment Manager

```
/usr/WebSphere6/AppServer/profiles/Dmgr01/bin/stopManager.sh
```

Example 11-11 Script to start the Deployment Manager

```
/usr/WebSphere6/AppServer/profiles/Dmgr01/bin/startManager.sh
```

2. Ensure that the HACMP services are active on HHOST2.

11.3.3 Testing Deployment Manager failover

A Deployment Manager failover does not affect client access to your application. We tested the following failover scenarios:

- Conducting a graceful failover test using the HACMP **takeover** option. Watch the SystemOut.log of the Deployment Manager for a successful start. Check all Node Agents to make sure they can synchronize before and after failover. There might be failed synchronization attempts during the failover. See Example 11-12 for a sample SystemOut.log of the Node Agent.

Example 11-12 Node Agent's SystemOut.log after Deployment Manager failover

```
#before failover
[4/13/05 12:19:36:632 CDT] 000001c0 NodeSyncTask  A   ADMS0003I: The
configuration synchronization completed successfully.
....
#during failover
[4/13/05 12:20:45:600 CDT] 000001c4 NodeSync      E   ADMS0015E: The
synchronization request cannot be completed because the node agent cannot
communicate with the deployment manager.
[4/13/05 12:20:45:615 CDT] 000001c4 NodeSyncTask  A   ADMS0016I: The
configuration synchronization failed.
....
#after failover
[4/13/05 12:23:56:548 CDT] 000001cb DiscoveryMBea I   ADMD0023I: The system
discovered process (name: dmgr, type: DeploymentManager, pid: 13478)
[4/13/05 12:24:01:732 CDT] 000001d0 NodeSyncTask  A   ADMS0003I: The
configuration synchronization completed successfully.
```

- Fall back to HHOST1 and conduct a shutdown failover test using **reboot –q** on HHOST1. Watch the SystemOut.log on the Deployment Manager and the Node Agents for success.

Other possible test scenarios include ending the Deployment Manager process using the **kill** command or unplugging the network cable of HHOST1.

11.4 Node Agent and application server

In our HACMP environment, we configured the test environment shown in Figure 11-3 on page 428 for Node Agent and application servers failover. The product binaries are installed on the shared file system. We assume that the Deployment Manager is installed, configured, and started on a remote system.

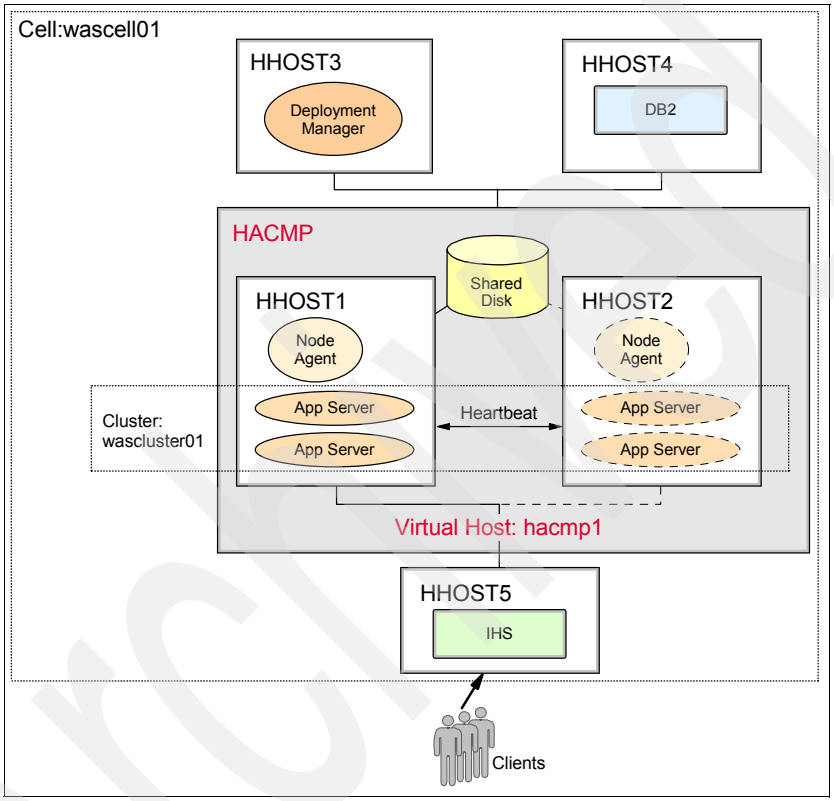


Figure 11-4 HACMP Node Agent and application server failover test scenario

11.4.1 Installing a Node Agent and application server or servers

To install a Node Agent and create a profile, do the following:

1. Start HACMP services on HHOST1 to mount the file system /usr/WebSphere6 from the shared disk array.
2. Install WebSphere Application Server Network Deployment V6.0 on HHOST1. The installation path is on the shared file system. In our example, the path is /usr/WebSphere6/AppServer.

3. If necessary, install Refresh packs or Fix packs. We installed the V6.0.1 Refresh pack.
4. Create a custom profile on HHOST1. The profile directory path is also on the shared file system, for example in our environment, the profile directory is /usr/WebSphere6/AppServer/profiles/Custom01. During profile creation, use the virtual host name (in our environment it is hacmp1) as the Host name, not the physical host name or physical IP address. See 9.4.2, “Installing WebSphere Application Server Network Deployment” on page 306 for more information.
5. Federate the node to the Deployment Manager while creating the custom profile or later using the **addNode** command. We federated our node to the Deployment Manager on HHOST3.
6. Create application servers on the node as desired. We have configured two application servers that are members of the same cluster (vertical scaling).

11.4.2 Configuring HACMP to run the Node Agents and application servers

To configure HACMP to run the Node Agents and application servers, you add the Node Agent stop and start commands as well as the appropriate start and stop commands for each application server in the failover unit to the HACMP stop and start scripts. Follow these steps:

1. Add the Node Agent and application server stop commands to the HACMP stop script (ha.stop) on both HHOST1 and HHOST2. See Example 11-13 where we added a Node Agent and two application servers (wasmember07 and wasmember08) to the script. Be aware that the application server names are case sensitive. Also make sure that you specify the correct profile directory.

Example 11-13 Script to stop application server node

```
/usr/WebSphere6/AppServer/profiles/Custom01/bin/stopServer.sh wasmember07
/usr/WebSphere6/AppServer/profiles/Custom01/bin/stopServer.sh wasmember08
/usr/WebSphere6/AppServer/profiles/Custom01/bin/stopNode.sh
```

2. Add the Node Agent and application server start commands to the HACMP start script (ha.start) on both HHOST1 and HHOST2. Notice that a **Sleep 60** command was added after the Node Agent start command. This is because the application servers depend on an active Node Agent to be able to start successfully. This value might need to be adjusted for your environment, depending on the time the Node Agent takes to start. See Example 11-14 on page 433.

Example 11-14 Script to start application server node

```
/usr/WebSphere6/AppServer/profiles/Custom01/bin/startNode.sh
sleep 60
/usr/WebSphere6/AppServer/profiles/Custom01/bin/startServer.sh wasmember07
/usr/WebSphere6/AppServer/profiles/Custom01/bin/startServer.sh wasmember08
```

3. Ensure that the HACMP services are active on HHOST2.

11.4.3 Testing Node Agent and application server failover

During the failover process, the application servers hosted on the HACMP node cannot serve any client requests. We tested the following two failover scenarios:

- Conducting a graceful failover test using the HACMP **takeover** option. Watch the SystemOut.log on the Node Agent and each application server for a successful start. Check the Node Agent to make sure it can synchronize with the Deployment Manager before and after failover. There might be failed synchronization attempts during the failover. See Example 11-15 and Example 11-16 on page 434 for sample SystemOut.logs of the Deployment Manager and Node Agent.
- Fall back to HHOST1 and conduct a shutdown failover test using **reboot –q** on HHOST1. Watch the SystemOut.log on the Node Agent and each application server for a successful start.

Example 11-15 Deployment Manager's SystemOut.log during node failover

```
#during failover
[4/15/05 18:11:57:750 CDT] 0000001c RmmPtpGroup   W   DCSV1111W: DCS Stack
DefaultCoreGroup at Member wascell01\wasdmgr01\dmgr: Suspected another member
because the outgoing connection from the other member was closed. Suspected
members is wascell01\wasna03\nodeagent. DCS logical channel is View|Ptp.
[4/15/05 18:11:57:753 CDT] 0000001c DiscoveryRmmP W   DCSV1111W: DCS Stack
DefaultCoreGroup at Member wascell01\wasdmgr01\dmgr: Suspected another member
because the outgoing connection from the other member was closed. Suspected
members is wascell01\wasna03\nodeagent. DCS logical channel is Connected|Ptp.
[4/15/05 18:11:57:756 CDT] 0000001c DiscoveryRmmP W   DCSV1113W: DCS Stack
DefaultCoreGroup at Member wascell01\wasdmgr01\dmgr: Suspected another member
because the outgoing connection to the other member was closed. Suspected
member is wascell01\wasna03\nodeagent. DCS logical channel is Connected|Ptp.
....
[4/15/05 18:12:33:981 CDT] 0000001c DiscoveryRmmP W   DCSV1111W: DCS Stack
DefaultCoreGroup at Member wascell01\wasdmgr01\dmgr: Suspected another member
because the outgoing connection from the other member was closed. Suspected
members is wascell01\wasna03\wasmember07. DCS logical channel is Connected|Ptp.
[4/15/05 18:13:09:341 CDT] 0000001c DiscoveryRmmP W   DCSV1111W: DCS Stack
DefaultCoreGroup at Member wascell01\wasdmgr01\dmgr: Suspected another member
```

```
because the outgoing connection from the other member was closed. Suspected
members is wascell01\wasna03\wasmember08. DCS logical channel is Connected|Ptp.
....
#after Node Agent started
[4/15/05 18:16:43:223 CDT] 0000001c MbuRmmAdapter I   DCSV1032I: DCS Stack
DefaultCoreGroup at Member wascell01\wasdmgr01\dmgr: Connected a defined member
wascell01\wasna03\nodeagent.
....
[4/15/05 18:16:53:790 CDT] 000015c6 DiscoveryMBea I   ADMD0023I: The system
discovered process (name: wasna03, type: NodeAgent, pid: 16008)
....
#after application server#1 started
[4/15/05 18:19:37:880 CDT] 0000001c MbuRmmAdapter I   DCSV1032I: DCS Stack
DefaultCoreGroup at Member wascell01\wasdmgr01\dmgr: Connected a defined member
wascell01\wasna03\wasmember07.
....
#after application server#2 started
[4/15/05 18:21:44:091 CDT] 0000001c MbuRmmAdapter I   DCSV1032I: DCS Stack
DefaultCoreGroup at Member wascell01\wasdmgr01\dmgr: Connected a defined member
wascell01\wasna03\wasmember08.
(complete failover)
```

Example 11-16 Node Agent's SystemOut.log during node failover

```
#after the Node Agent started
[4/15/05 18:15:52:038 CDT] 0000000a WsServerImpl  A   WSVR0001I: Server
nodeagent open for e-business
[4/15/05 18:15:55:208 CDT] 0000002d NodeSyncTask  A   ADMS0003I: The
configuration synchronization completed successfully.
....
[4/15/05 18:18:50:199 CDT] 0000003a DiscoveryMBea I   ADMD0023I: The system
discovered process (name: wasmember07, type: ManagedProcess, pid: 25444)
....
[4/15/05 18:20:56:052 CDT] 00000056 DiscoveryMBea I   ADMD0023I: The system
discovered process (name: wasmember08, type: ManagedProcess, pid: 30534)
```

11.4.4 Application with embedded messaging failover

The WebSphere setup for this scenario is the same as for the Node Agent and application server failover scenario described previously. The Deployment Manager has been installed on a remote system and the custom profile is installed into the HACMP cluster system. In addition, for this scenario, we installed an application that puts and gets messages to/from embedded messaging destinations.

For this scenario, you need to make sure that the messaging engine (ME) runs only on the application server that is running in the HACMP cluster. If you want to

operate the ME in a specific application server, you have to configure the proper core group policy for the ME. (For more information about core group policy configurations, see 6.2.4, “Core group policy” on page 188.) Therefore, we configured a static policy and relating match criteria as well as static group servers. We specified a certain messaging engine in the match criteria and selected the application server running in the HACMP cluster as the static group server. Thus the messaging engine can now only run in our failover application server.

Testing the message engine failover scenario

First you must put messages on the destination. You can confirm the queued messages in the Administrative Console. Select **Service integration** → **Buses** → ***Bus_Name*** → **Messaging engines** → ***Messaging_Engine_Name*** → **Queue points** → ***Queue_Point_Identifier*** → **Runtime tab** → **Messages**. See Figure 11-5 for an example of this panel.

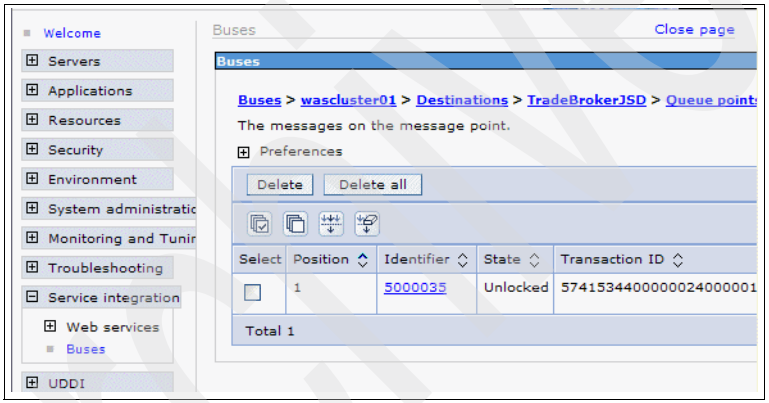


Figure 11-5 Confirm queued messages

After putting messages on the destination, conduct a shutdown failover test using the **`reboot -q`** option. After the failover, the application should be able to process the messages properly. Watch the logs or database on HHOST2 to see if the ME was started successfully and is indeed processing the messages.

You can confirm that the ME is starting properly in the application server's SystemOut.log. See Example 11-17 for the messages that are written to the SystemOut.log.

Example 11-17 Application server's SystemOut.log during failover

```
[4/19/05 17:42:33:608 CDT] 0000002b SibMessage    I
[wascluster01:wascluster01.004-wascluster01] CWSID0016I: Messaging engine
wascluster01.004-wascluster01 is in state Joined.
....
[4/19/05 17:42:33:803 CDT] 00000030 SibMessage    I
[wascluster01:wascluster01.004-wascluster01] CWSID0016I: Messaging engine
wascluster01.004-wascluster01 is in state Starting.
[4/19/05 17:42:38:453 CDT] 00000030 SibMessage    I
[wascluster01:wascluster01.004-wascluster01] CWSIS1538I: The messaging engine
is attempting to obtain an exclusive lock on the data store.
[4/19/05 17:42:38:558 CDT] 00000032 SibMessage    I
[wascluster01:wascluster01.004-wascluster01] CWSIS1537I: The messaging engine
has acquired an exclusive lock on the data store.
....
[4/19/05 17:42:42:883 CDT] 00000030 SibMessage    I
[wascluster01:wascluster01.004-wascluster01] CWSIP0212I: messaging engine
wascluster01.004-wascluster01 on bus wascluster01 is starting to reconcile the
WCCM destination and link configuration.
[4/19/05 17:42:42:941 CDT] 00000030 SibMessage    I
[wascluster01:wascluster01.004-wascluster01] CWSIP0213I: messaging engine
wascluster01.004-wascluster01 on bus wascluster01 has finished reconciling the
WCCM destination and link configuration.
[4/19/05 17:42:43:146 CDT] 00000030 SibMessage    I
[wascluster01:wascluster01.004-wascluster01] CWSID0016I: Messaging engine
wascluster01.004-wascluster01 is in state Started.
....
```

11.5 Transaction Manager failover with No Operation policy

In this scenario, we want to control the Transaction Manager (TM) failover by using HACMP and the No Operation (NoOP) policy. HACMP detects the hardware or software failure and executes the Transaction Manager failover.

A general discussion on configuring the Transaction Manager failover with the No Operation Policy is provided in 9.6, "Transaction Manager failover with No Operation policy" on page 313.

To take advantage of the Transaction Manager peer recovery in HAManager, you need to use a shared file system which all application servers that can perform transaction log recovery can access at the same time. This shared file system then holds the transaction logs for all these application servers. So we had to configure another test environment where we used an NAS disk system. See Figure 11-6. As you can see, in this test environment both nodes can access the shared file system. We mounted the shared file system to the same mount point (/mnt/nas) on both nodes.

The HACMP setup itself is basically the same as for the Deployment Manager failover or Node Agent and application server failover, for details see “Configuration basics of HACMP” on page 420 and “Managing resources” on page 423. However, you need to perform a different WebSphere installation and some additional configuration steps, and you also need to change the previously used HACMP start and stop scripts. The sections that follow describe the necessary changes.

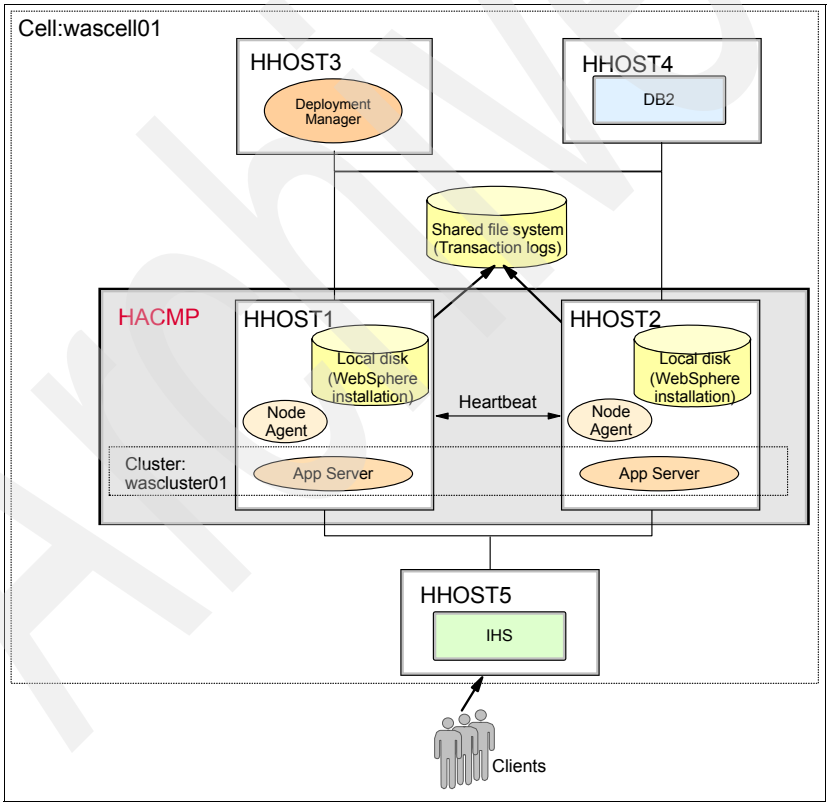


Figure 11-6 HACMP Transaction Manager failover test environment

11.5.1 WebSphere configuration

You need to install the product binaries and the profiles locally instead of on the shared disks for this scenario. You also need to enable high availability for the Transaction Manager and move the transaction log directories onto the shared disk. Figure 11-7 shows our WebSphere setup.

The WebSphere configuration steps are described in great detail in 9.6.3, “Configuring WebSphere for TM No Operation policy” on page 317. This is what you need to do:

1. Follow the steps described for configuring the TM in “Enabling high availability for the Transaction Manager” on page 317.
2. Copy the transaction logs to the shared file system as described in “Copying existing transaction logs to the shared directory” on page 318.
3. Inform the Transaction Manager of the new transaction log locations as described in “Pointing the TM to the new location for the transaction logs” on page 319.
4. Create the HA policy. See “Creating the No Operation policy for the Transaction Manager” on page 320.

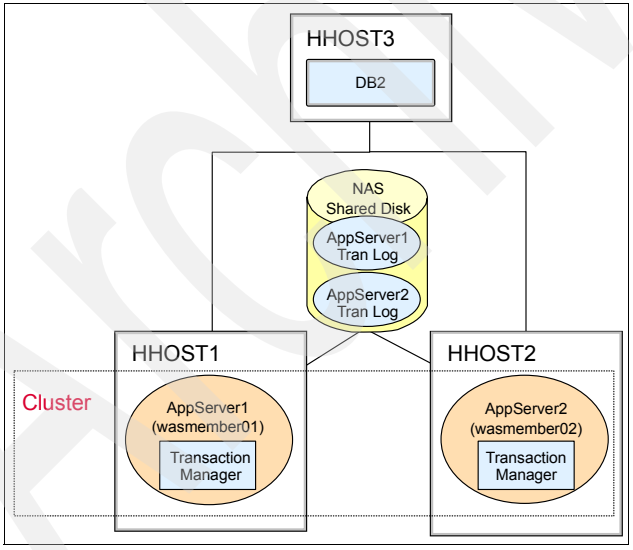


Figure 11-7 WebSphere configuration overview

11.5.2 HACMP configuration

The concept of what needs to be configured on the cluster software side is described in 9.6.4, “Configuring external clustering software for Transaction Manager No Operation policy recovery” on page 325. Here we describe how this needs to be done for HACMP.

There is however a difference in the HACMP setup compared to the other clustering software setups: Our HACMP is an Active/Passive setup while the other clustering software setups are Active/Active. Because of this different setup, the application server needs to be started outside of HACMP and only the failover of the Transaction Manager is done using HACMP.

You need to configure new start and stop scripts in HACMP as well as for starting the application servers outside of HACMP. These start and stop scripts call the wasctrl-as and wasctrl-tm scripts described in “Scripts to start, stop, and monitor WebSphere resources” on page 331.

Tip: For AIX V5.1, you might have to install an additional module (perl.libwww-5.41.0.0.exe) for the Perl program to execute the wasctrl-tm script. If you use AIX 5.2 or 5.3, this module is installed by default.

This example assumes that you run the TM for wasmember01 on HHOST1 and want to failover the TM to wasmember02 on HHOST2.

1. First you need a script to start the application server wasmember01 and its Transaction Manager (which is configured with the No Operation policy) on the local system (HHOST1). This script is called server.start and is shown in Example 11-18. The server.start script calls the wasctrl-as and wasctrl-tm scripts with the value `start` as the action parameter.

   Note: You need to execute this script on a command line to start the application server and TM outside of HACMP.

Example 11-18 server.start script on HHOST1 - starting the application server and TM

```
#!/bin/ksh
/usr/bin/HAScrp/wasctrl-as start /usr/WebSphere/AppServer/profiles/Custom01/
9080 wasmember01
/usr/bin/HAScrp/wasctrl-tm start /usr/WebSphere/AppServer/profiles/Custom01/
9080 HHOST1 8879 IBM_hc=wascluster01,type=WAS_TRANSACTIONS wascell01
wasmember01 /usr/bin/HAScrp/ wasna05
```

The same action needs to be done on HHOST2 to start wasmember02. See Example 11-19 for the server.start script for the second system.

Example 11-19 server.start script on HHOST2 - starting the application server and TM

```
#!/bin/ksh
/usr/bin/HAScrp/wasctrl-as start /usr/WebSphere/AppServer/profiles/Custom01/
9080 wasmember02
/usr/bin/HAScrp/wasctrl-tm start /usr/WebSphere/AppServer/profiles/Custom01/
9080 HHOST2 8879 IBM_hc=wascluster01,type=WAS_TRANSACTIONS wascell01
wasmember02 /usr/bin/HAScrp/ wasna06
```

2. The application server (wasmember01) is monitored in HACMP on HHOST1 using the ha.monitor script shown in Example 11-20.

Example 11-20 ha.monitor script on HHOST1

```
#!/bin/ksh
/usr/bin/HAScrp/wasctrl-as status  /usr/WebSphere/AppServer/profiles/Custom01/
9080 wasmember01
RC=$?
exit ${RC}
```

3. You need to define the HACMP start script (ha.start). This script is shown in Example 11-21 and is executed by HACMP on HHOST2 in case of a failure of wasmember01. This script starts the TM of wasmember01 on the secondary system (which is wasmember02 on HHOST2).

Example 11-21 ha.start script - TM failover to HHOST2

```
#!/bin/ksh
/usr/bin/HAScrp/wasctrl-tm start /usr/WebSphere/AppServer/profiles/Custom01/
9080 HHOST2 8879 IBM_hc=wascluster01,type=WAS_TRANSACTIONS wascell01
wasmember01 /usr/bin/HAScrp/ wasna05
```

4. The HACMP stop script for HHOST1 is shown in Example 11-22. This script is executed on HHOST1 when monitoring detected that wasmember01 is down and thus the failover of the TM to wasmember02 is initiated. The script calls the wasctrl-as and wasctrl-tm scripts with the value `stop` as the action parameter.

Example 11-22 ha.stop script on HHOST1

```
#!/bin/ksh
/usr/bin/HAScrp/wasctrl-tm stop /usr/WebSphere/AppServer/profiles/Custom01/
9080 HHOST1 8879 IBM_hc=wascluster01,type=WAS_TRANSACTIONS wascell01
wasmember01 /usr/bin/HAActivationScripts/ wasna05
/usr/bin/HAScrp/wasctrl-as stop /usr/WebSphere/AppServer/profiles/Custom01/
9080 wasmember01
```

If you also want to setup failover of the Transaction Manager for wasmember02, then you need to define another HACMP resource group and configure similar scripts to start the other cluster members' TM.

11.5.3 Testing Transaction Manager with NoOP policy failover

In our environment, we killed the process ID of the application server and shut down the system:

1. Ensure HACMP services are active on both HHOST1 and HHOST2.
2. Conduct a failover test by ending the application server process using the **`kill -9`** ***`Application_server_process_ID`*** command on HHOST1. You can find the PID of the application server by looking at the *application_server_name*.pid file in the application server's log directory.
3. Verify the Transaction Manager's failover:
   a. In the Administrative Console click **Servers** → **Core groups** → **Core group settings** → ***Core_group_name***. Select the Runtime tab.
   b. Enter `type=WAS_TRANSACTIONS` into the Group name properties field, and then click **Show groups**. A panel similar to the one in Figure 11-8 is displayed. This panel shows all Transaction Manager High availability groups of your environment.

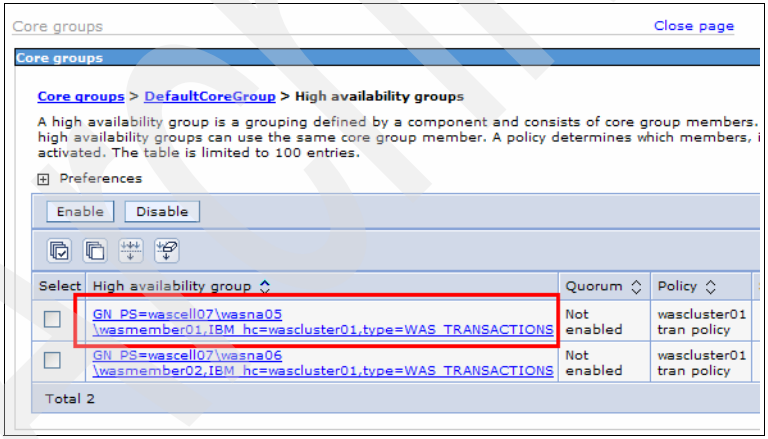


Figure 11-8 Checking Transaction Manager failover - 1

c. Select the High availability group which is related to the failed application server's Transaction Manager. The panel in Figure 11-9 is shown. Then verify that the Transaction Manager's Status is active and the Transaction Manager is running on the other application server.

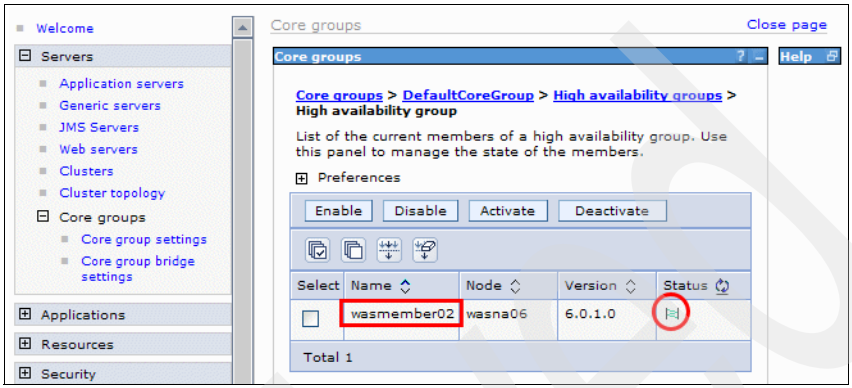


Figure 11-9 Checking Transaction Manager failover - 2

If there are any in-doubt transactions when the failover happens, you would see transaction recovery messages in the SystemOut.log of the take-over application server.

11.6 Summary

There are various ways to set up the HACMP failover behavior, for example, one system can be the backup for several other primary systems. In our environment, a simple cascading resource group was set up for demonstrating how WebSphere Application Server leverages the HACMP features to provide a highly available environment.

The WebSphere Application Server software is installed on the disk array shared by the primary and standby machines. When the primary machine fails, the WebSphere Application Server configuration on the disk array will be mounted to the standby machine. The HACMP start script on the standby machine then runs each command in the script to start all necessary servers. Also, the standby machine takes over the service adapter of the primary machine. During this failover process, WebSphere is not available for its clients. There is a need to add error recovery logic into the client program to handle the server failure.

In addition to testing the WebSphere Application Server administrative servers (Deployment Manager and Node Agent), we tested the failover of application servers with 2-phase Commit transactions and active messaging engines to

make sure that potential in-doubt transactions are recovered and unprocessed messages are processed after the failover occurs.

The adjustments of several settings given in this paper are based on our lab environment. As with performance tuning, the values might vary for different business environments.

11.7 Reference

- HACMP for AIX 5L Web site:

  http://www.ibm.com/servers/aix/products/ibmsw/high_avail_network/hacmp.html

- HACMP documentation:

  http://www.ibm.com/servers/eserver/pseries/library/hacmp_docs.html

- *MQSeries for AIX - Implementing with HACMP Version 2.0* guide:

  ftp://ftp.software.ibm.com/software/integration/support/supportpacs/individual/mc63.pdf

- *IBM DB2 Universal Database Enterprise Edition for AIX and HACMP/ES* guide:

  ftp://ftp.software.ibm.com/software/data/pubs/papers/db2ee-aixhacmp.pdf

12

WebSphere and VERITAS Cluster Server

This chapter provides an introduction into the basics of clustering IBM WebSphere Application Server V6 using VERITAS Cluster Server (VCS). We show how to configure VCS to manage WebSphere V6 as well how to set up WebSphere Application Server to leverage the benefits a VCS provides.

Important:

- Throughout this chapter you find information and material from VERITAS Software Corporation. This information was printed by permission of VERITAS Software Corporation.
- At the time of writing this redbook, VCS was not in the list of supported software. There are however three different kinds of configurations: Supported, Not supported and Other. VCS falls into the *Other* category. For details about how IBM Support handles this category, see the document *IBM WebSphere Application Server - Clarification of configurations support*, which is available at:

  http://www.ibm.com/support/docview.wss?rs=180&context=SSEQTP&uid=swg27004311

  For the latest list of supported products see:

  http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html

12.1 Introduction to VCS

Setting up a cluster using VCS requires a basic understanding of the way VCS works and what its main components are. It is especially important to understand the model of service and resource groups. This understanding is necessary for the successful configuration of a VERITAS cluster that provides high availability to applications such as WebSphere Application Server. We also discuss the way VCS monitors and manages resources using agents.

12.1.1 How VERITAS Cluster Server works

VERITAS Cluster Server acts like an abstraction layer over a group of physical systems which typically behave like one single system. It can also monitor and control applications residing on those systems. You can configure VCS to run your application on one ore more systems at the same time.

A cluster consists of hardware or software resources. Examples for resources are disks and disk groups, network interfaces and IP addresses, or any application residing on the cluster. Those resources can be managed by the VCS. VCS will start, stop, and monitor resources, which allows you to logically group resources and to failover to another system when a critical resource fails.

To control the cluster communication and membership, VCS uses:

- High-Availability Daemon (HAD) which runs on all systems in the cluster. The HAD maintains the status of all resources in the cluster.
- Low Latency Transport (LLT) is used for communication between the cluster systems. LLT balances the communication stream between the assigned network interfaces. Additionally, it is used for the heartbeat, which is a health check to determine the Group Membership Service (GAB).
- Group Membership Service (GAB) works together with LLT to maintain the cluster communication and cluster membership. It ensures the message delivery to the cluster systems.

12.1.2 Configuration basics of VCS

VCS uses two files for its configuration. These files contain the configuration of the resources and service groups, their properties and dependencies:

- The main.cf file for the cluster
- The types.cf file for the different resource types

You can create those files in the following ways:

- Using the Cluster Manager GUI (available as a Java or Web Console)
- Using the command line interface
- Using an editor, such as vi, emacs, or Notepad while VCS is stopped

12.1.3 Managing resources

Management of resources means starting, stopping and monitoring them. In some cases there are dependencies between resources. For example to unmount a file system, the application that uses the file system has to be taken offline before unmounting the file system. To simplify the management of a set of resources and resource dependencies, you can define a service group. You can distinguish between three different types of service groups:

- *Failover Service Group* means that the group runs only on one system at a time.
- *Parallel Service Group* means that the group runs concurrently on the systems in the cluster.
- *Hybrid Service Group* means a combination of the Failover and Parallel Service Group used for replicated clusters.

More information about service groups is available at:

`http://www.veritas.com`

Managing resources depends on the type of the resource. VCS comes with a bundle of resource types, for example Application, DNS, Mount, or Volume. As you add a resource to your service group, you choose a resource type and select whether it is critical or just enabled. Critical means, a failure of this resource initiates a failover of the service group. Enabled means that this resource is monitored by an agent.

Agents are management processes for predefined resource types or third party applications, for example, IBM WebSphere MQ or IBM DB2 UDB. They consist of type declaration files and binaries to enlarge the management capabilities of VCS.

The concept of service groups allows VCS to distinguish between application and node failures. Service groups are switched between the nodes in case of a failover. Therefore, it is necessary to use a *Virtual IP Address (VIP)*, which is also moved when failing over. The use of virtual IP addresses is also known as *IP Aliasing*.

12.1.4 Using Cluster Agent for IBM WebSphere MQ

VERITAS Cluster Server provides so-called Enterprise Agents for different third party applications. The Enterprise Agent for IBM WebSphere MQ provides a more extended failure detection than the Bundled Agents, for example:

- *Process check* looks for the appropriate process of a specified Queue Manager.
- *Second-Level Check* pings the Queue Manager using a MQClient, which ensures that the process is really working.

Important: Enterprise Agents usually need a separate license. Contact VERITAS for detailed information.

To perform the installation, for example on UNIX or Linux, follow the instructions provided in the *VERITAS Cluster Server Agent 3.5 for WebSphere MQ Installation and Configuration Guide* found at:

`http://ftp.support.veritas.com/pub/support/products/ClusterServer_UNIX/273084.pdf`

After the agent is installed, you have to import the VProWSMQTypes.cf file. Then you are able to create and configure a IBM WebSphere MQ Queue Manager resource.

12.1.5 Using Cluster Agent for IBM DB2 UDB

VERITAS Cluster Server provides so-called Enterprise Agents for different third party applications. The Enterprise Agent for IBM DB2 UDB provides a more extended failure detection than the Bundled Agents, for example:

- *In depth monitoring* allows checking the output of **`db2nps`** (shows active processes of the instance).
- Also possible is an evaluation of the output of **`db2gcf`** (shows, for example, the current status of an instance).

Important: Enterprise Agents usually need a separate license. Contact VERITAS for detailed information.

To install this agent, for example on UNIX/Linux, follow the instructions provided in *VERITAS Cluster Server Enterprise Agent 4.0 for DB2 Installation and Configuration Guide*, which is available at:

http://ftp.support.veritas.com/pub/support/products/ClusterServer_UNIX/270374.pdf

After the agent is installed, you have to import the DB2udbTypes.cf file. Then, you are able to create and configure a service group for IBM DB2 UDB.

12.2 Planning and preparation

These are the prerequisites to follow our scenario:

- We assume you have at least two systems running with VCS. Both systems share one storage device.
- We further assume that you have checked the prerequisites needed for IBM WebSphere Application Server Network Deployment V6. These are found at:

  http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html
- You have an IP Alias for the Deployment Manager/Node Agent system.

  It is also possible to install without having a separate IP Alias. In this case, you have to join or link to an existing service group that uses an IP Alias. Make sure that this configuration fits into your systems architecture respectively meets your non-functional requirements.
- You should determine how you are going to configure WebSphere. Specifically, if you are going to separate the binary files from the configuration files. You have two options: you can put the binaries and configuration files on the shared disk or you can put the binaries on a local disk and the configuration files on the shared disk.

12.3 Deployment Manager

In IBM WebSphere V6, the Deployment Manager is a key component in a WebSphere cell. It provides the administrative interface, keeps the master repository, and synchronizes the configuration with all nodes in the cell. For more information about the Deployment Manager see *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451. In WebSphere V6 the Deployment Manager is not required for runtime transactions, but needed for administration and configuration changes. For that reason it might be useful to make the Deployment Manager highly available. One way to do this, is using VCS with shared disks.

12.3.1 Installing the Deployment Manager

To install the Deployment Manager follow the steps provided in 9.3, “Deployment Manager high availability” on page 298. This section covers the various options for how to install the Deployment Manager so that the configuration information is available to each node in the cluster.

Installing IBM WebSphere Application Server Network Deployment V6 with an IP failover based cluster software requires that your host name is bound to your IP Alias (Virtual IP Address). When creating the Deployment Manager profile, the host name associated with the virtual IP address is used in the configuration. Whenever the Deployment Manager is accessed, the host name associated with the VIP is used. For example, when specifying the URL for the Administrative Console, use

```
http://viphostname:9060/ibm/console/
```

The benefit of using the host name that is associated with the VIP is that it isolates the client from the specific system that is currently hosting the Deployment Manager. If System A is hosting the Deployment Manager and it fails over to System B, then the clustering software switches the virtual IP address to be associated with System B. However, the DNS name (and thus the URL for the Administrative Console) does not need to change and will continue to work.

Figure 12-1 on page 451 shows our lab setup, which is as follows:

- We have two systems that have access to a Network Attached Storage (NAS) device. The systems utilize iSCSI to attach to the NAS which provides simultaneous and concurrent access to the directories that contain the WebSphere configuration data.
- We configured VCS to have an Active/Passive configuration which means that the Deployment Manager is running only on one of the systems at a time. One system is deemed the primary system and when the Deployment Manager process fails on that system, the secondary system becomes active and all requests of the Deployment Manager occur on the secondary system.
- We installed the WebSphere V6 binaries locally on both systems.
- We created the Deployment Manager profile on the shared file system located on the NAS. This makes the configuration data available to both the primary and secondary system.

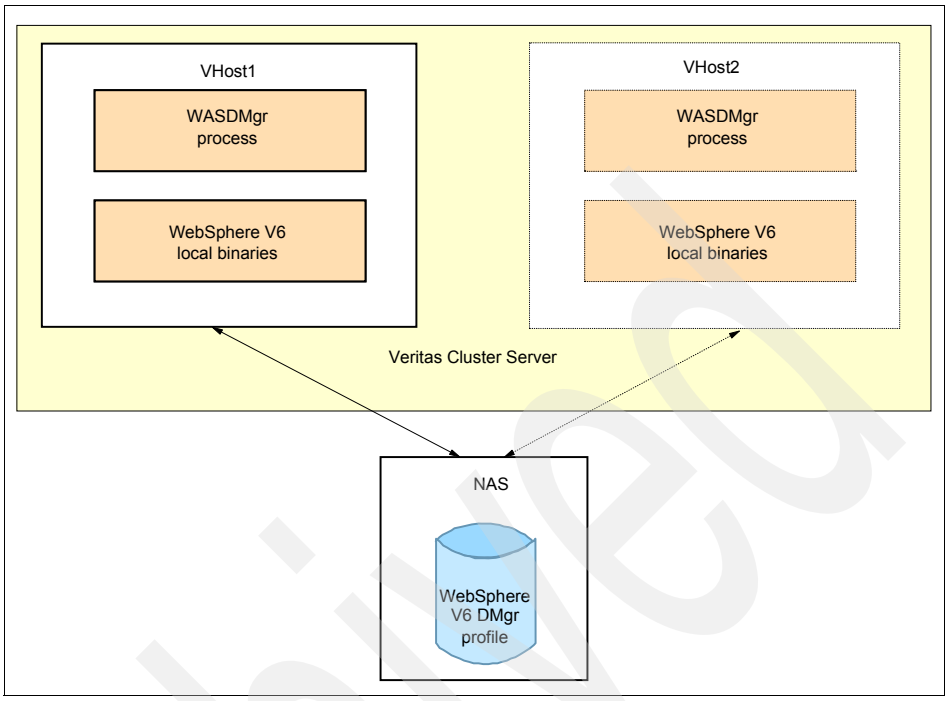


Figure 12-1 Lab example showing two systems connected to NAS

12.3.2 Configuring VCS to run the Deployment Manager

To leverage the benefits of VCS, you have two configuration options:

- Setting up a unique service group for the Deployment Manager failover.

  This method requires a unique virtual IP Address for every Deployment Manager Service Group. So the failing over of a Deployment Manager Service Group is disjoint from another Deployment Manager Service Group failover.

- Using the existing ClusterService Service Group

  VCS creates a default service group called ClusterService. The ClusterService service group contains a resource for the network interface card (NIC), for the virtual IP address, and for the VCSWeb applications. It is easy to add the mount resource and application resource for the Deployment Manager to this service group. However, resources that are by default critical in this service group might cause the Deployment Manager to failover (even though the Deployment Manager is not dependent on the resources).

Individual circumstances will help determine which option is appropriate for you. With everything equal, it appears that the first option, and the ability to create a Deployment Manager Service Group - with the resource required for that Deployment Manager encapsulated in the service group - is the better option.

We assume that you are familiar with the administration of VCS. So we highlight only a few important points. VCS provides the *application* resource type that we used to represent the Deployment Manager. The application resource requires a start and stop program to control an application. The application resource also requires at least one of the following for monitoring the application:

- A monitor program
- A pid file
- A process name displayed by the **ps** command

In our example, we provided the start and stop scripts under the <*WAS_HOME*>/bin directory and use the pid file for monitoring.

When the Deployment Manager server starts up, it creates a pid file called dmgr.pid. This file is located in the <*WAS_HOME*>/profiles/*your_profile_name*/logs/dmgr/ directory and contains the process ID of the Deployment Manager. The application resource for the Deployment Manager reads that file and looks for the given process ID to monitor this process. If this process is not active, then a failover is initiated.

In our example, we use an NAS shared file system to contain the profile for the Deployment Manager. Therefore, in VCS, we need to configure a *mount* resource for mounting/unmounting the file system that contains the profile for the Deployment Manager.

The procedure we followed to make the Deployment Manager highly available using VCS is as follows:

1. Create a new service group.

   See Figure 12-12 on page 463 for the configuration panel. Ensure that you select **Failover** for the Service Group Type. This indicates to VCS that it is an Active/Passive service group. When a critical resource on the primary system fails, then VCS will failover all resources in the service group to the secondary system.

   Notice that the systems named Bottechia and SystemB are the two systems in our environment. The system called Bottechia has a priority of 0 and the system called SystemB has a priority of 1. In our example, Bottechia is considered the primary system and SystemB the secondary system.

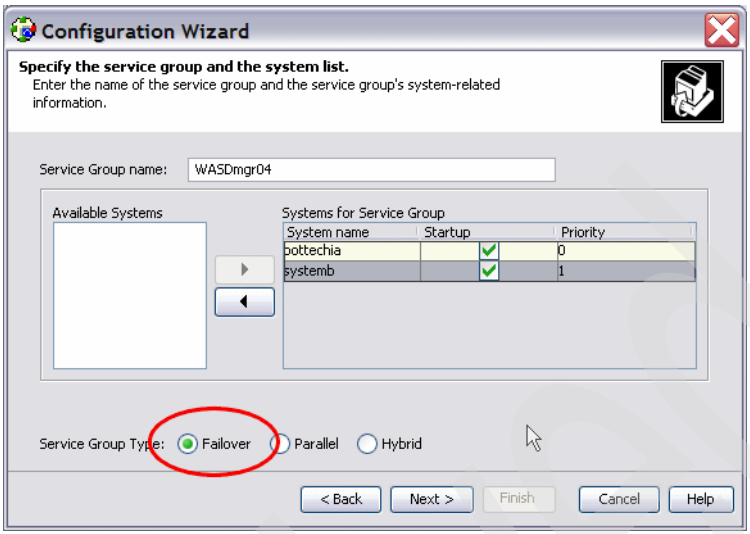


Figure 12-2 Creating a new Service Group using the Configuration Wizard for DMgr

2. Add resources to that service group:

   a. Add a resource type Application for the Deployment Manager. See Figure 12-13 on page 466.

      In our example, we used the following WebSphere commands as the start and stop programs for the Deployment Manager application resource:

      ```
      <WAS_HOME>/profiles/dmgrprofile/bin/startManager
      <WAS_HOME>/profiles/dmgrprofile/bin/stopManager
      ```

      We used the pid file as the monitoring option for the resource. The pid file is located at:

      <*WAS_HOME*>/profiles/*dmgrprofile*/logs/dmgr/dmgr.pid

      The use of the pid file for monitoring the Deployment Manager is very good in detecting a crash of the Deployment Manager's JVM.

      Attention: If you plan to enable global security, you need to add a user and a password to your soap.client.props or sas.client.props file. Which props file to use depends on the protocol you use to connect to the Deployment Manager. For more information about this, see *WebSphere Application Server V6: Security Handbook*, SG24-6316.

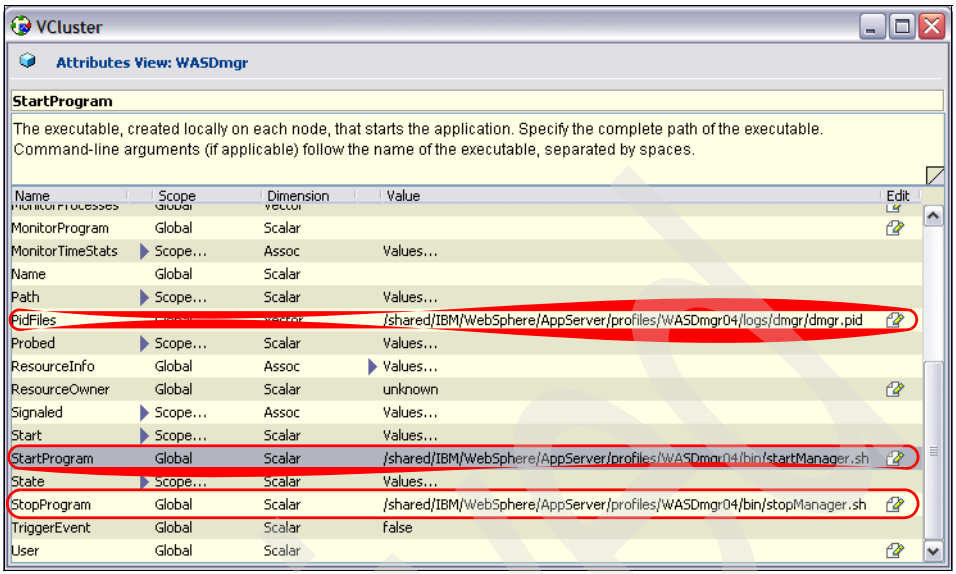


Figure 12-3 Attributes of our resource WASDMgr

b. Add a resource type Mount for the shared disk that contains the shared WAS profile.

   In our example, the configuration files managed by the Deployment Manager reside on a shared file system with a mount point name of /shared. This resource ensures that the file system is mounted and available to the Deployment Manager. See Figure 12-14 on page 467.

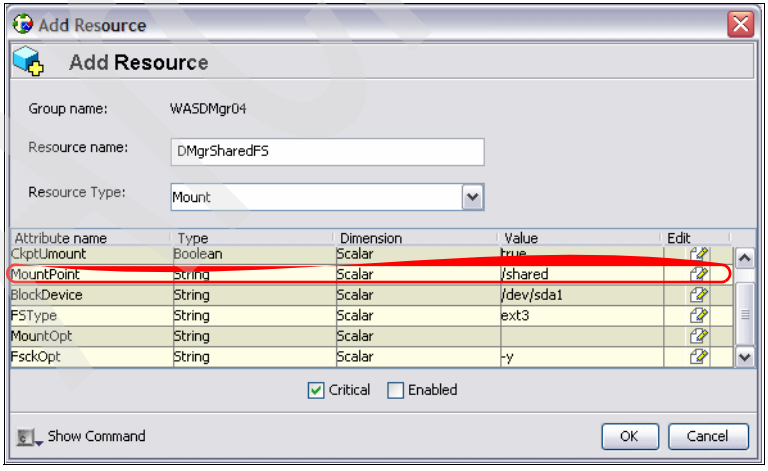


Figure 12-4 Attributes of resource DMgrSharedFS

Important: At this point, you have to create the resource for the VIP for the Deployment Manager.

Due to some restrictions in our lab environment, we were not able to obtain a VIP for the Deployment Manager and therefore, we used the default IP for this scenario. However, the configuration of a separate VIP is described in 12.4.2, “Configuring VCS to run the Node Agents and application server or servers” on page 457 as we were able to use a VIP for the Node Agent and application server scenario.

c. Set resource dependencies.

Figure 12-5 shows the resource dependencies necessary when adding our WASDMgr resources to the existing ClusterService service group.

In our example, we configured a start sequence:

i. Check if NIC (network card interface) is up.
ii. Assigning the VIP (Virtual IP Address) to that NIC.
iii. Mounting the shared file system.
iv. Starting the Deployment Manager.

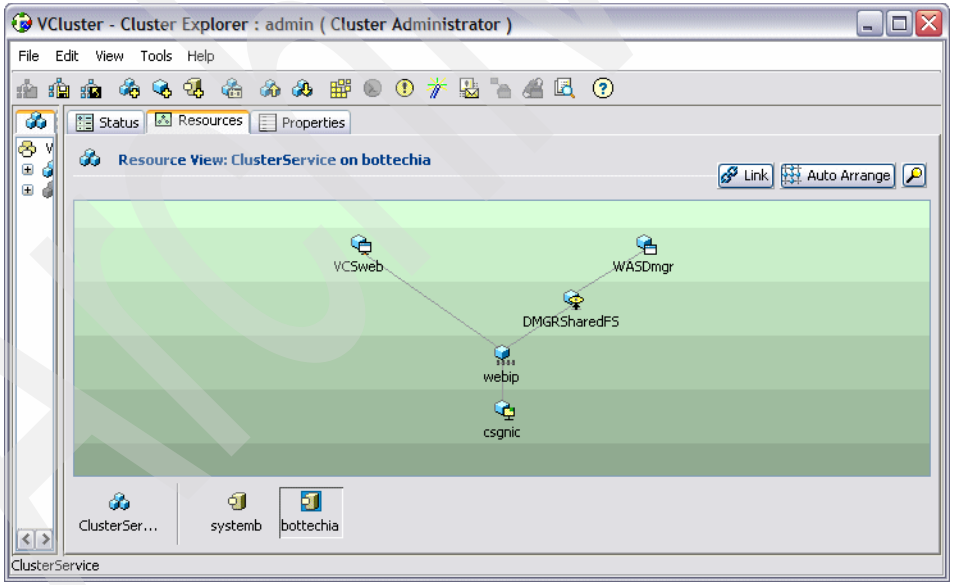


Figure 12-5 Resource dependencies - Deployment Manager configuration

12.3.3 Testing Deployment Manager failover

Now that you configured VCS to manage the Deployment Manager, we recommend to test if everything works as expected. Three possible tests you could do, are:

1. Shutting down the system where the Deployment Manager is running.
2. Killing the Deployment Manager process using the `kill -9` command (on UNIX or Linux).
3. Deleting the pid.file.

In all of these test scenarios, VCS as we set it up realized the failure and switched the Deployment Manager to the second system.

These tests are just examples of possible test scenarios. You can try any other test that might better fulfill your needs.

12.4 Node Agent and application server

The need to make the Node Agent highly available is reduced in WebSphere Version 6. In 9.4, “Node Agent and application server high availability” on page 304, we explained why and how to make the Node Agent highly available with clustering software in a generic way. This section focuses on how to make the Node Agent highly available using VCS.

12.4.1 Installing a Node Agent and application server or servers

For information about how to install a Node Agent or application server see 9.4, “Node Agent and application server high availability” on page 304.

Figure 12-6 on page 457 shows the configuration of our test environment. As described for the Deployment Manager, we split up our installation: the binaries are located locally and the profiles are placed on the shared disk drive. In our case the shared disk is an NAS that can be concurrently accessed by both systems.

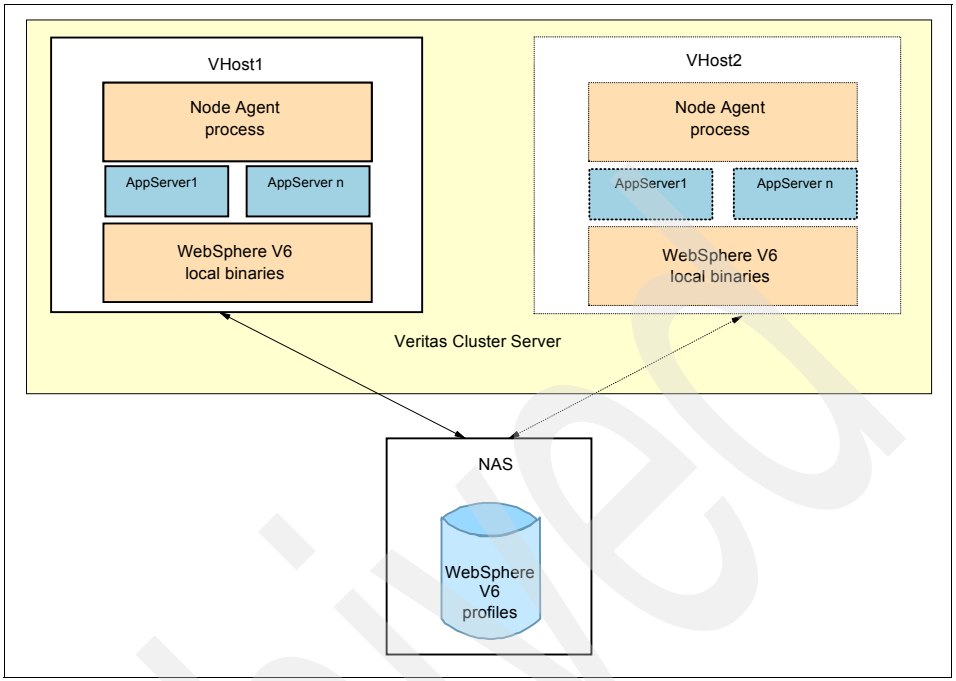


Figure 12-6 Lab example showing two systems connected to NAS

12.4.2 Configuring VCS to run the Node Agents and application server or servers

One key concept in configuring the highly available Node Agent is that the Node Agent is coupled with all the application servers that reside in the node. If the Node Agent is to failover, all the corresponding application servers must failover also. The ramifications of this concept for configuring VCS is that you must define a single service group that contains the resources for the Node Agent application and all the application servers that reside in the node. Then this service group will failover as a group to maintain the coupling that is needed.

In our example, we configured the Node Agent and application servers as application resources. As mentioned before, VCS application resources require a start and stop program to control an application. In our example, we provided the start and stop scripts under the <*WAS_HOME*>/bin directory and the pid file for monitoring.

To monitor application resources you need to provide at least one of the following:

- A monitor program
- A pid file
- A process name displayed by the **ps** command

At start, the Node Agent creates a pid file which is found in this location:

<*WAS_HOME*>/profiles/*your_profile_name*/logs/nodeagent/nodeagent.pid

This pid file contains the process ID of the Node Agent. The VCS application resource reads the pid file and looks for the process to determine if the Node Agent is up or down. In case the process is down, a failover is initiated.

In our example, we use an NAS shared file system to contain the profile for the Node Agent and the application servers. Therefore, in VCS, we need to configure a *mount* resource for mounting or unmounting the file system that contains the profiles. The procedure we followed to make the Node Agent and corresponding application servers highly available using VCS is as follows:

1. Create a new Service Group

   Ensure that you select the **Failover** option for the Service Group Type. This indicates to VCS that it is an Active/Passive service group. When a critical resource on the primary system fails, then VCS will failover all the resources in the service group to the secondary system. Notice that the systems named Bottechia and SystemB are the two systems in our lab. The system called Bottechia has a priority of 0 and the system called SystemB has the priority of 1. In our example, Bottechia is considered the primary system and SystemB the secondary system. See Figure 12-7.

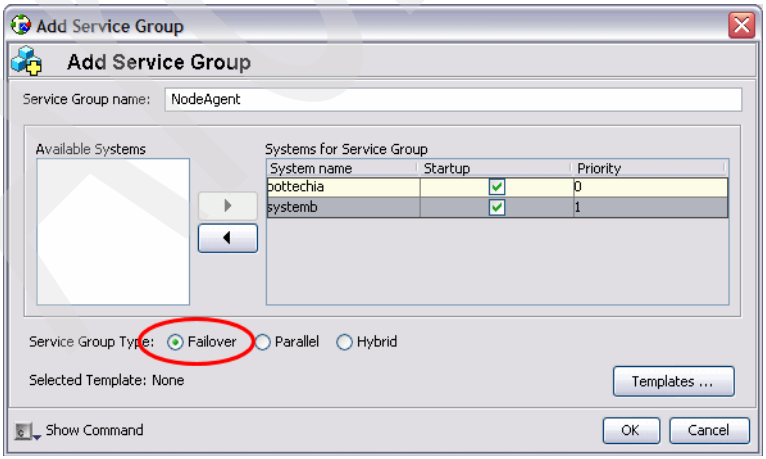


Figure 12-7 Creating a new Service Group for Node Agent

2. Add resources to that service group:

   a. Add a resource type Application for the Node Agent.

      In our example, we used the following WebSphere commands as the start and stop programs for the Node Agent application resource:

      ```
      <WAS_HOME>/profiles/testprofile/bin/startNode
      <WAS_HOME>/profiles/testprofile/bin/stopNode
      ```

      We used the pid file as the monitoring option for the resource. The pid file is located at:

      <*WAS_HOME*>/profiles/*testprofile*/logs/nodeagent/nodeagent.pid

      The use of the pid file for monitoring the Node Agent is very good in detecting a crash of the Node Agents's JVM. See Figure 12-8 on page 460 for our configuration.

      Attention: If you plan to enable global security, you need to add a user and a password to your soap.client.props or sas.client.props file. Which props file to use depends on the protocol you use to connect to the Deployment Manager. For more information about this, see *WebSphere Application Server V6: Security Handbook*, SG24-6316.

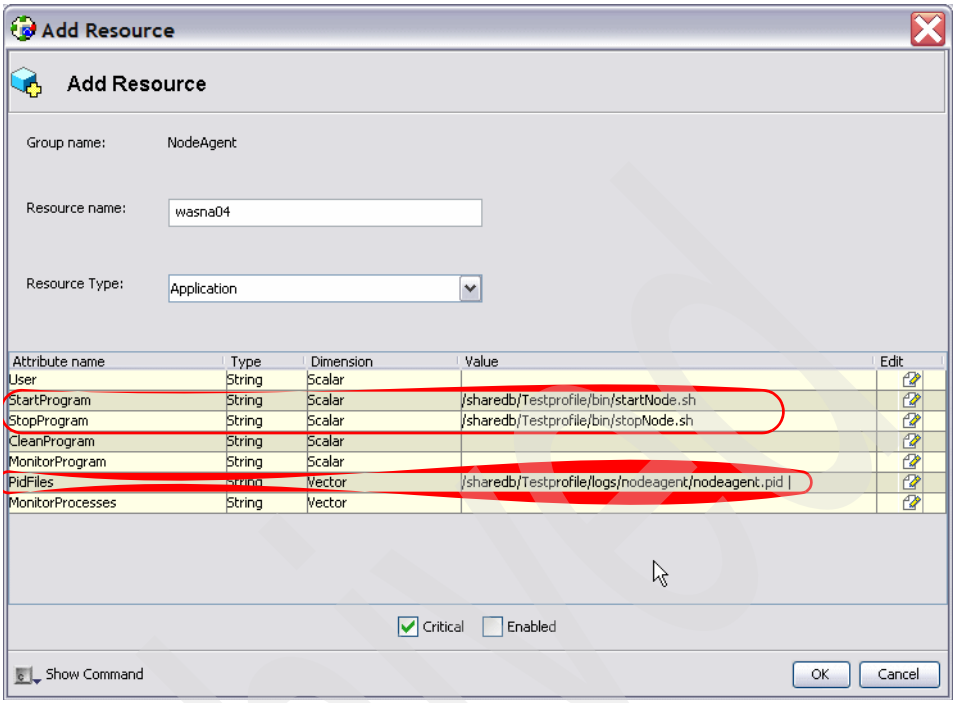


Figure 12-8 Attributes of resource NodeAgent

b. Add a resource type Mount for the shared disk that has to be mounted.

In our example, the configuration files managed by the Node Agent reside on a shared file system with a mount point name /sharedB. This resource ensures that the file system is mounted and available to the Node Agent. See Figure 12-9 on page 461.

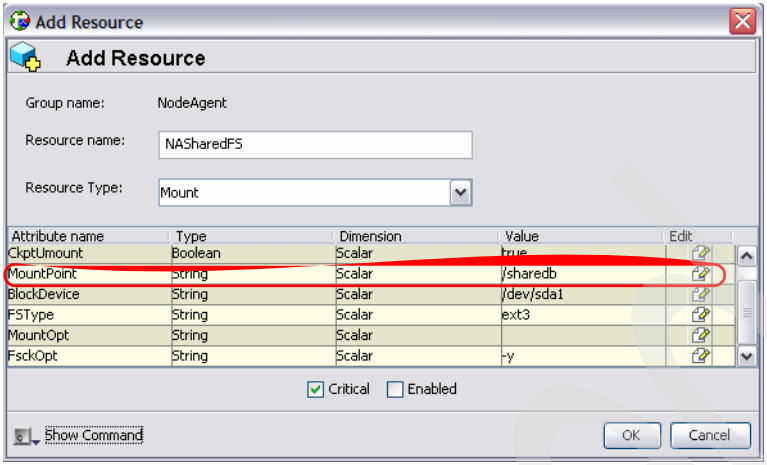


Figure 12-9 Attributes of resource NASharedFS

c. Add a resource for the IP Alias as shown in Figure 12-10.

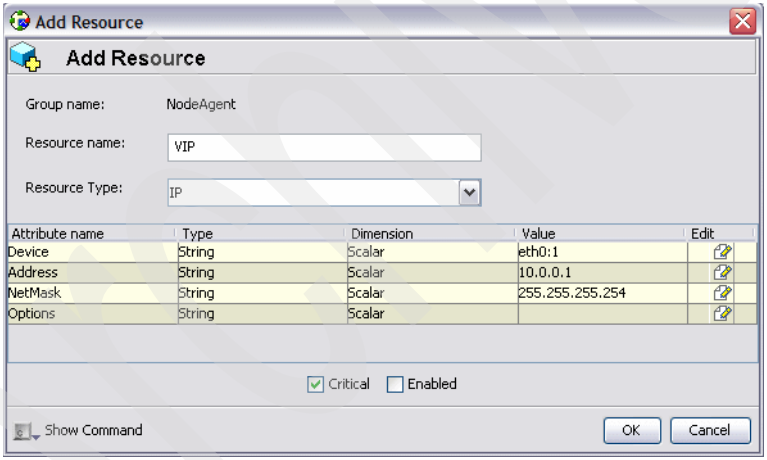


Figure 12-10 Attributes of resource IP Alias

d. Add resources for each application server on the node.

Because each application server must failover with the Node Agent, a resource is defined for each application server. In our example, we used the WebSphere commands as the start and stop programs for the Node Agent application resource:

```
<WAS_HOME/>profiles/testprofile/bin/startServer
<WAS_HOME>/profiles/testprofile/bin/stopServer
```

We used the pid file as the monitoring option for the resource. The pid file is located at:

<*WAS_HOME*>/profiles/*testprofile*/logs/AppServerName/server.pid

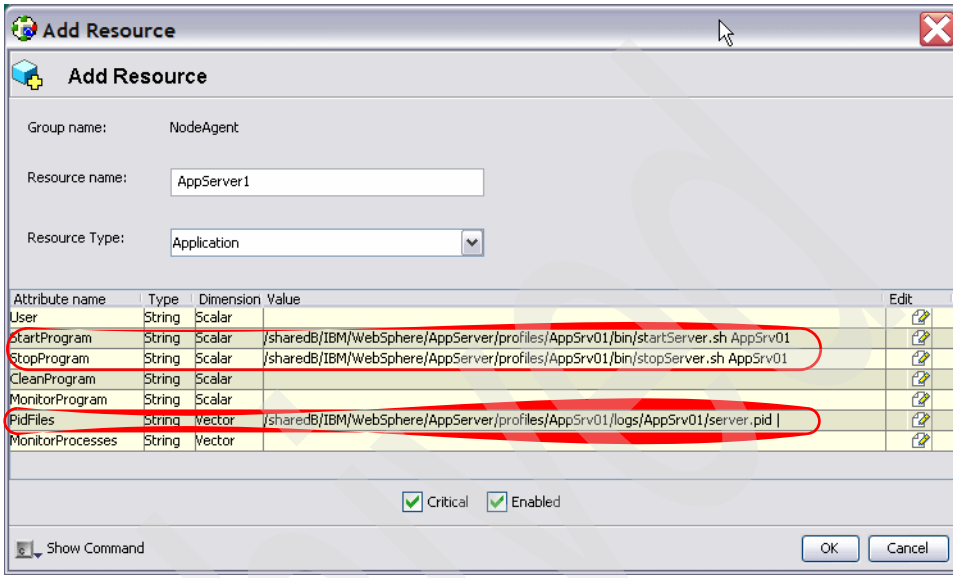


Figure 12-11 Resource for application server AppSrv01

e. Set resource dependencies

Figure 12-12 on page 463 shows the resource dependencies necessary when adding our Node Agents' resources to the Node Agent service group.

In our example, we configured a start sequence:

i. Assigning VIP (Virtual IP Address).
ii. Mounting the shared file system.
iii. Starting the Node Agent.
iv. Start each application server.

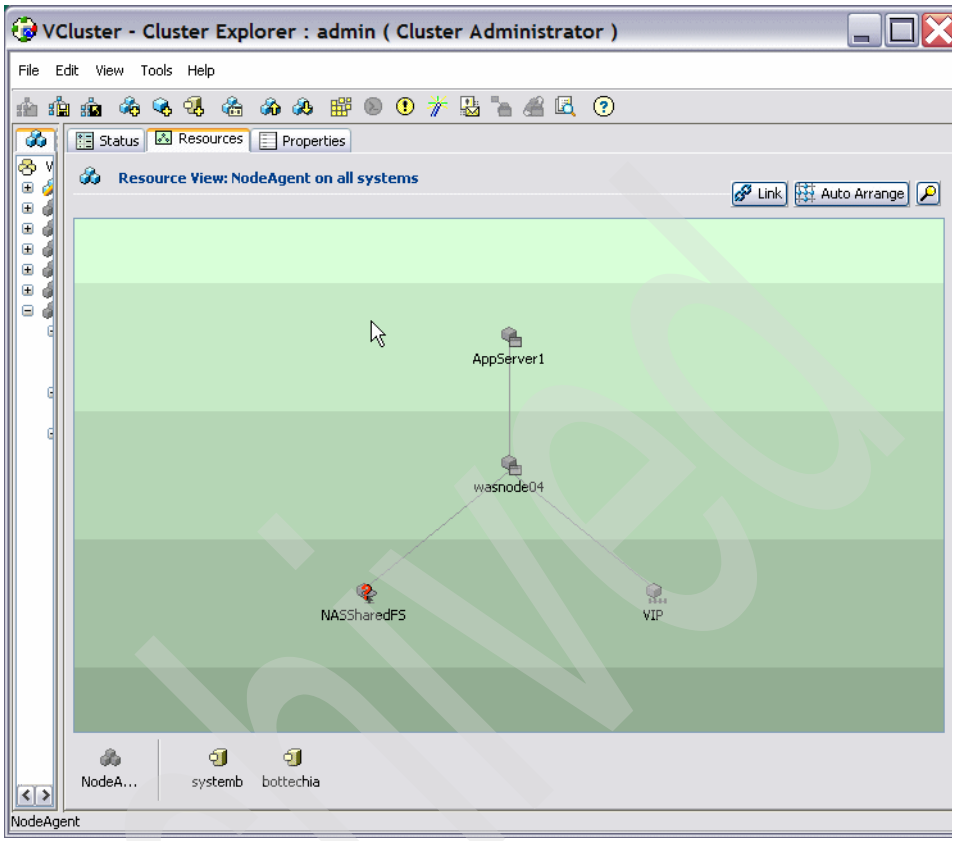


Figure 12-12 Resource dependencies - Node Agent configuration

12.4.3 Testing Node Agent and application server failover

Now that you configured VCS to manage the Node Agent, we recommend to test if everything works as expected. You can perform the following possible tests:

1. Shutting down the system where the Node Agent is running.
2. Killing the Node Agent process using the **`kill -9`** command (on UNIX or Linux).
3. Deleting the pid.file.

In our setup, for all of these test scenarios, VCS realized the failure and switched the Node Agent to the second system.

These tests are just examples of possible test scenarios. You can try any other test that might better fulfill your needs.

12.5 Transaction Manager failover with No Operation policy

A general discussion of configuring the Transaction Manager failover with the No Operation policy is provided in 9.6, “Transaction Manager failover with No Operation policy” on page 313. In this section we discuss how to configure VCS to correctly failover the Transaction Manager with the No Operation (NoOP) recovery policy.

In order to describe the configuration we refer to the example defined in “Transaction Manager with No Operation policy scenario” on page 316. In our setup, we have two WebSphere Application Server nodes. One node is the system named Bottechia and the other is the system named SystemB. Each node hosts a single application server: TradeServer1 on Bottechia and TradeServer2 on SystemB. Both application servers are cluster members of TradeCluster.

For this scenario, we need to define the following service groups in VCS:

- WAS servers service group

  This group will be configured as parallel (Active/Active) and will start, stop, and monitor the Node Agent and the cluster members.

- TM activate for system Bottechia

  This group will be configured as failover (Active/Passive) and will activate, deactivate and monitor the TM on system Bottechia (primary) and failover to system SystemB (secondary).

- TM activate for system SystemB

  This group will be configured as failover (Active/Passive) and will activate, deactivate and monitor the TM on system SystemB (primary) and failover to system Bottechia (secondary).

We also need to configure links (dependencies) between each group. The TM activate groups are linked to the WAS servers group. This means that the TM activate groups will not start until the WAS servers group completes starting. The TM activate groups are not linked to each other.

12.5.1 WebSphere configuration

Follow the instructions in 9.6.3, “Configuring WebSphere for TM No Operation policy” on page 317 to configure WebSphere for the transaction log failover.

12.5.2 VCS configuration: service groups and resources

Next, we describe each service group and the resources that make up each group.

WAS servers service group

This group must be configured as a parallel group. This means that the resources start on each system simultaneously and there is no failover. This group contains two application resources and a mount resource. The application resources are responsible for starting, stopping and monitoring the Node Agents and the cluster members. The mount resource is responsible for mounting the shared file system that contains the transaction logs.

Here is a description of the resources that you need for this service group:

- Node Agent application resource

  This resource is responsible for starting, stopping, and monitoring the Node Agent. There is a definition for each system (Bottechia and SystemB) so that the resource can start the Node Agent associated with each system simultaneously. In our example, this resource is marked as enabled but not as critical. This is because the application server can still process transactions if the Node Agent fails.

  In this resource, we used the WebSphere commands of **`startNode`** and **`stopNode`** to start and stop the Node Agent. We used the nodeagent.pid file as the monitoring technique. VCS reads the PID in this file and monitors it to ensure that the Node Agent is available.

  Figure 12-13 on page 466 shows our configuration.

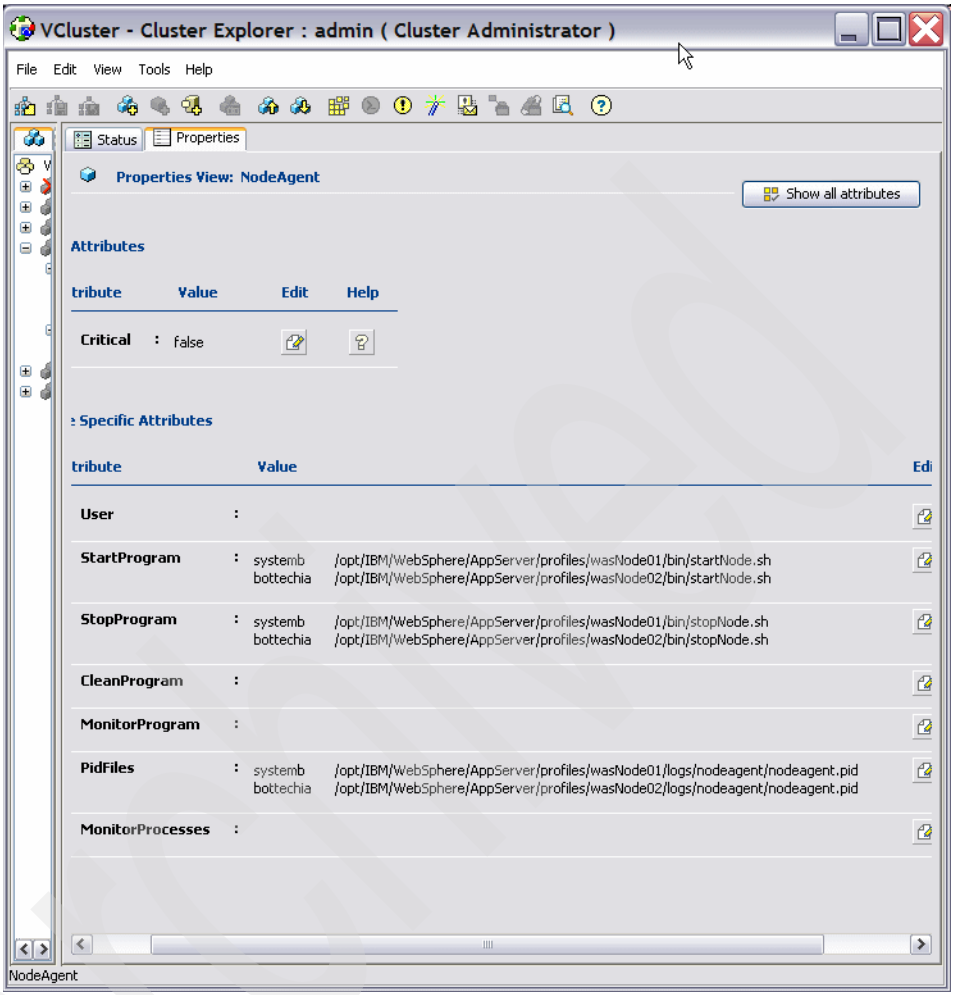


Figure 12-13 Node Agent resource for TM NoOP policy

- Cluster member application resource

  This resource is responsible for starting, stopping and monitoring the cluster members. See Figure 12-14 on page 467. There is a definition for each system (Bottechia and SystemB) so that the resource can start the cluster members associated with each system simultaneously. In our example, this resource is marked as enabled and critical.

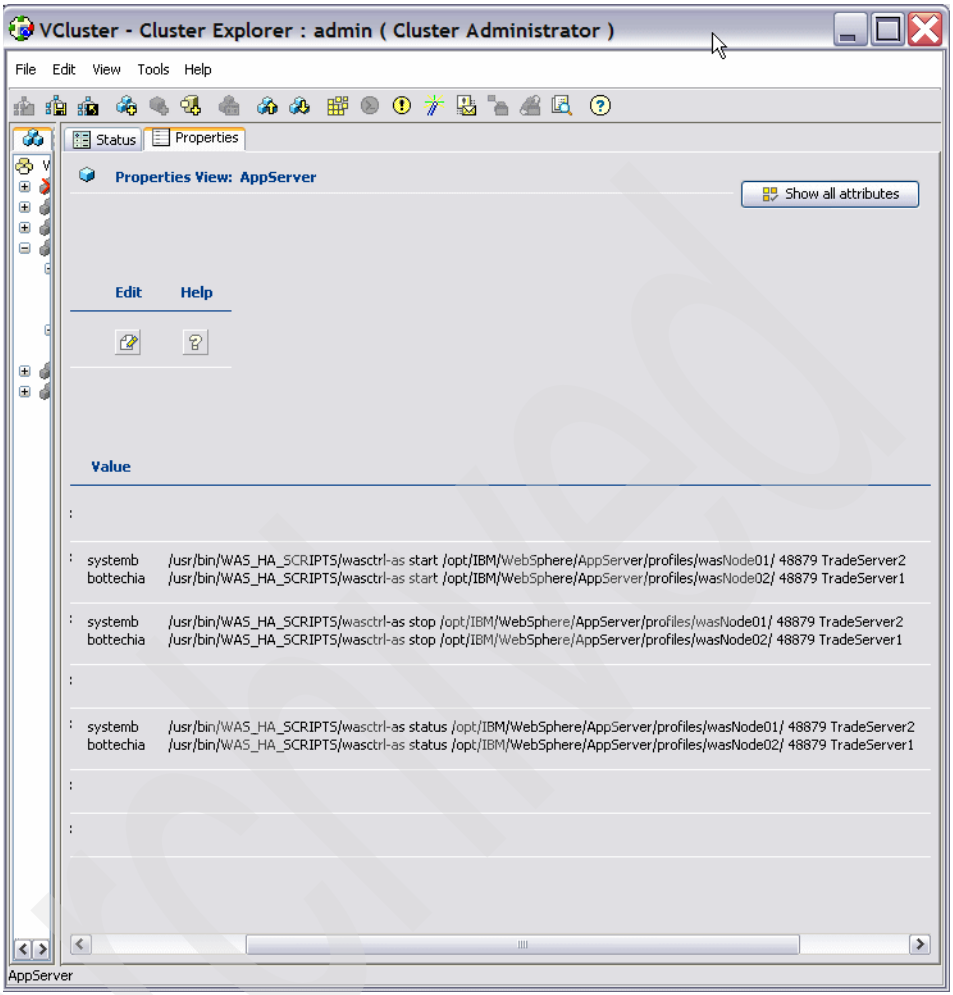


Figure 12-14 Cluster member resource for TM NoOP policy

As shown in Figure 12-14, this resource uses the wasctrl-as script that is described in “Scripts to start, stop, and monitor WebSphere resources” on page 331 to start, stop, and monitor the cluster members.

The script is needed because the cluster members halt during start and wait for the HAManager to be notified to activate the Transaction Manager for the cluster member. The script starts the cluster members with a time-out setting so that they return after a certain amount of time. This allows the TM activate groups to activate the Transaction Manager for this cluster member. Notice that the cluster member TradeServer2 is started on SystemB and TradeServer1 is started on Bottechia.

Note: The wasctrl-as and wasctrl-tm scripts have to be modified slightly. The scripts set return codes for the status of the resource. VCS has defined ranges of return codes to indicate whether a resource status is online, offline or unknown. The scripts have to be modified so that the correct status value is returned for VCS. The scripts need to have the following values modified:

- UNKNOWN=0
- ONLINE=110
- OFFLINE=100

- Shared file system mount resource

  This resource mounts the shared file system that contains the transaction logs. This resource could also be located in the TM activate groups, but in our example it is in the WebSphere Application Servers servers group. See Figure 12-15 on page 469.

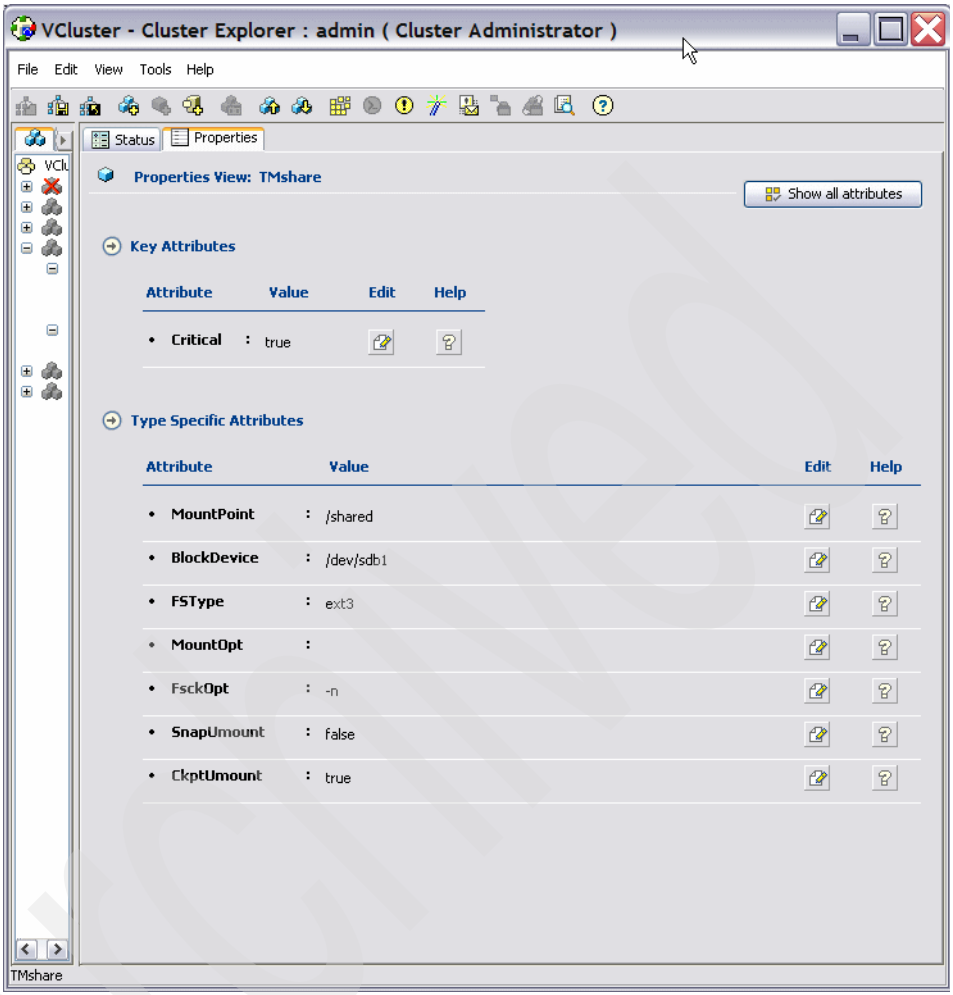


Figure 12-15 Mount resource for TM NoOP policy

The dependencies between the resources in this group are shown in Figure 12-16 on page 470. The cluster members depend on the Node Agent and mount point resources. The Node Agent must be started first for the cluster members to start successfully.

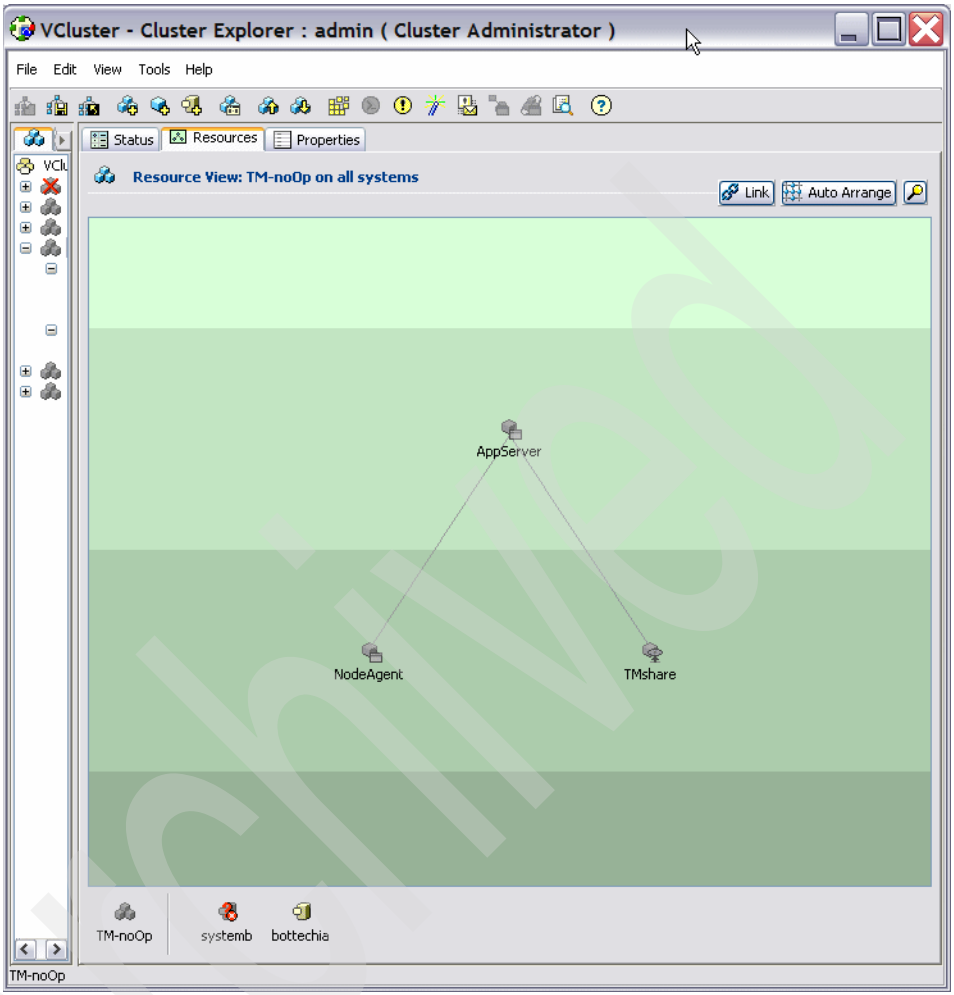


Figure 12-16 WAS servers service group dependencies for TM NoOP policy

TM activate for system Bottechia

This group is configured as a failover group, with the primary system being Bottechia and the secondary system being SystemB. In our example, we configured this group with one application resource. It is likely that this group would be configured with a mount resource for the shared file system that contains the transaction logs. This is a likely scenario in a customer environment, but, in our example, we mounted the shared file system with the servers service group.

- Activate TM application resource

  This resource activates, deactivates and monitors the Transaction Manager and is shown in Figure 12-17 on page 472. For the primary system (Bottechia), this resource activates the Transaction Manager and allows the cluster member on this system to continue its start processing and then to log transactions for requests made to the cluster member.

  In a failure situation, the activate indicates to the Transaction Manager of the cluster member on the secondary system (SystemB) to start peer recovery of the transactions that are in the primary cluster member's recovery log.

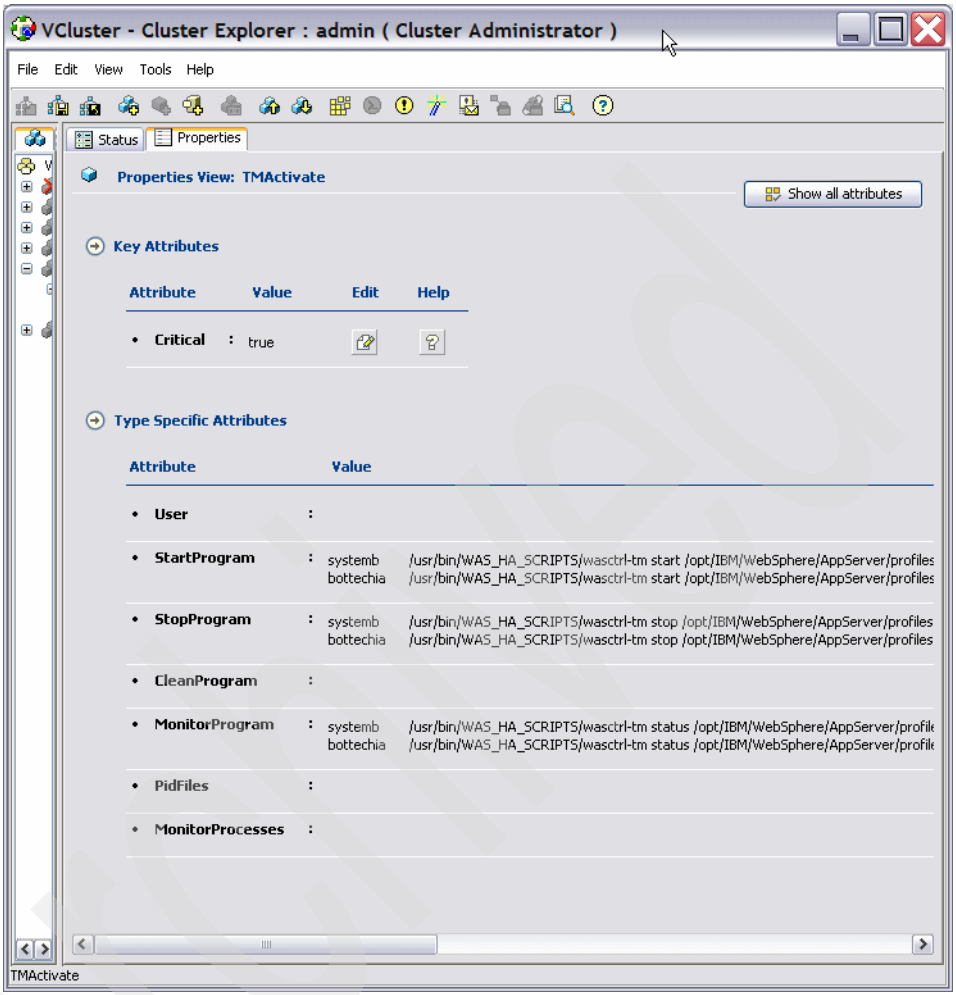


Figure 12-17 TM activate resource for TM NoOP policy

This resource specifies the script wasctrl-tm as explained in “Scripts to start, stop, and monitor WebSphere resources” on page 331. It is important to understand the invocation parameters for the scripts on the different systems (primary and secondary). The script invocation for activating the TM on the *primary* system is:

```
/usr/bin/WAS_HA_SCRIPTS/wasctrl-tm start
/opt/IBM/WebSphere/AppServer/profiles/wasNode02/ 49080 bottechia 48879
IBM_hc=TradeCluster,type=WAS_TRANSACTIONS WASCell04 TradeServer1
/usr/bin/WAS_HA_SCRIPTS/ bottechiaNode01
```

The first parameter is the action and indicates to start the TM. The second parameter is the installation directory of the node. The next two parameters are the host and port for the servlet that is installed on the cluster member to perform the MBean actions. The next port is the **wsadmin** binding port. When activating against a cluster member that is not started (the primary cluster member), the script uses a JACL script and this is the SOAP port for running **wsadmin** and the JACL script. The next parameter is part of the match criteria for selecting which TM to work against. The fully qualified match criteria consists of the cell name, node name, server name and the name/value pairs for IBM_hc and type. These are the 6th, 7th, 8th and 10th parameters. The 9th parameter is a pointer to where the scripts are installed.

The invocation for activating the TM recovery on the *secondary* system is:

```
/usr/bin/WAS_HA_SCRIPTS/wasctrl-tm start
/opt/IBM/WebSphere/AppServer/profiles/wasNode01/ 49080 systemb 48879
IBM_hc=TradeCluster,type=WAS_TRANSACTIONS WASCell04 TradeServer1
/usr/bin/WAS_HA_SCRIPTS/ bottechiaNode01
```

Notice that the secondary invocation indicates to run on the secondary system (SystemB) but to activate the TM for the primary system (Bottechia). The parameters for the node install directory, the servlet port, host name and SOAP port are all pointing to the secondary system (even though some of the values are the same as for the primary system). The parameters that deal with the match criteria (name/value pairs, cell name, node name and server name) are all pointing to the TM associated with the primary cluster member (Bottechia).

TM activate for system SystemB

This group is very similar to the previous group. This group is configured as a failover group, with the primary system being SystemB and the secondary system being Bottechia. In our example, we configured this group with one application resource.

- Activate TM application resource

  This resource activates, deactivates and monitors the Transaction Manager. For the primary system (SystemB), this resource activates the Transaction Manager and allows the cluster member on this system to continue its start processing and to log transaction for requests made to the cluster member.

  In a failure situation, the activate indicates to the Transaction Manager of the cluster member on the secondary system (Bottechia) to start peer recovery of the transactions that are in the primary cluster member's recovery log.

  This resource specifies the script wasctrl-tm as explained in "Scripts to start, stop, and monitor WebSphere resources" on page 331. It is important to understand the invocation parameters for the scripts on the different systems

(primary and secondary). The script invocation for activating the TM on the *primary* system is:

```
/usr/bin/WAS_HA_SCRIPTS/wasctrl-tm start
/opt/IBM/WebSphere/AppServer/profiles/wasNode01/ 49080 systemb 48879
IBM_hc=TradeCluster,type=WAS_TRANSACTIONS WASCell04 TradeServer2
/usr/bin/WAS_HA_SCRIPTS/ systembNode01
```

The primary invocation indicates to run on the primary system (SystemB), and activate the TM for the primary system (SystemB). The parameters for the node install directory, the servlet port, host name and SOAP port are all pointing to the primary system. The parameters that deal with the match criteria (name/value pairs, cell name, node name and server name) all are pointing to the TM associated with the primary cluster member (SystemB).

The invocation for activating the TM recovery on the *secondary* system is:

```
/usr/bin/WAS_HA_SCRIPTS/wasctrl-tm start
/opt/IBM/WebSphere/AppServer/profiles/wasNode02/ 49080 bottechia 48879
IBM_hc=TradeCluster,type=WAS_TRANSACTIONS WASCell04 TradeServer2
/usr/bin/WAS_HA_SCRIPTS/ systembNode01
```

Notice that the secondary invocation indicates to run on the secondary system (Bottechia) but to activate the TM for the primary system (SystemB). The parameters for the node install directory, the servlet port, host name and SOAP port are all pointing to the secondary system (even though some of the values are the same as the primary system). The parameters that deal with the match criteria (name/value pairs, cell name, node name and server name) all are pointing to the TM associated with the primary cluster member (SystemB).

12.5.3 Testing Transaction Manager with NoOP policy failover

When you have finished the setup, you should test if your configured environment performs failover as expected. Possible tests include killing the process ID of the application server and shutting down the system.

Refer to 11.5.3, “Testing Transaction Manager with NoOP policy failover” on page 441 in the HACMP chapter for instructions on how to verify whether your Transaction Manager failed over as expected.

Alternatively, you can look at the messages in the SystemOut.log file of the failed-over server where you will find messages indicating the start.

12.6 Default messaging provider failover with No Operation policy

A general discussion of configuring messaging engine failover with the No Operation Policy is provided in 9.7, “Default messaging provider failover with No Operation policy” on page 347. In this section we discuss how to configure VCS to correctly failover the messaging engine with the No Operation policy.

In order to describe the configuration we refer to the example defined in 9.7.2, “Default messaging provider with No Operation policy scenario” on page 348. In our setup, we have two WebSphere Application Server nodes. One node is the system named Bottechia and the other is the system named SystemB. Each node hosts a single application server which is part of a cluster called TradeCluster. The cluster members are TradeServer1 on Bottechia and TradeServer2 on SystemB. The Trade 6 application is configured with two messaging engines (ME): TradeCluster.000-TradeCluster and TradeCluster.001-TradeCluster. The primary cluster member for the TradeCluster.000-TradeCluster messaging engine is TradeServer1 and the primary cluster member for the TradeCluster.001-TradeCluster messaging engine is TradeServer2.

In order to configure VCS, we need to define the following service groups:

1. WAS servers service group

   This group will be configured as parallel (Active/Active) and will start, stop, and monitor the Node Agent and the cluster members.

2. ME activate for system Bottechia

   This group will be configured as failover (Active/Passive) and will activate, deactivate and monitor the ME on system Bottechia (primary) and failover to system SystemB (secondary).

3. ME activate for system SystemB

   This group will be configured as failover (Active/Passive) and will activate, deactivate and monitor the ME on system SystemB (primary) and failover to system Bottechia (secondary).

We also need to configure links (dependencies) between each group. The ME activate groups are linked to the WAS servers group. This means that the ME activate groups will not start until the WAS servers group completes starting. The ME activate groups are not linked to each other.

12.6.1 WebSphere configuration

Follow the instructions in 9.7.3, “Configuring WebSphere for default messaging provider No Operation policy” on page 349 to configure WebSphere for this scenario.

12.6.2 VCS configuration: service groups and resources

This section describes each service group and the resources that make up each group.

WAS servers service group

This is configured in exactly the same way that the TM NoOP Policy WAS servers service group is configured. Refer to 12.5, “Transaction Manager failover with No Operation policy” on page 464.

ME activate for system Bottechia

This group is configured as a failover group, with the primary system being Bottechia and the secondary system being SystemB. In our example, we configured this group with one application resource:

- Activate ME application resource

  This resource activates, deactivates and monitors the messaging engine. It is shown in Figure 12-18 on page 477. For the primary system (Bottechia), this resource activates the messaging engine. In a failure situation, the messaging engine on the cluster member on the secondary system (SystemB) is activated.

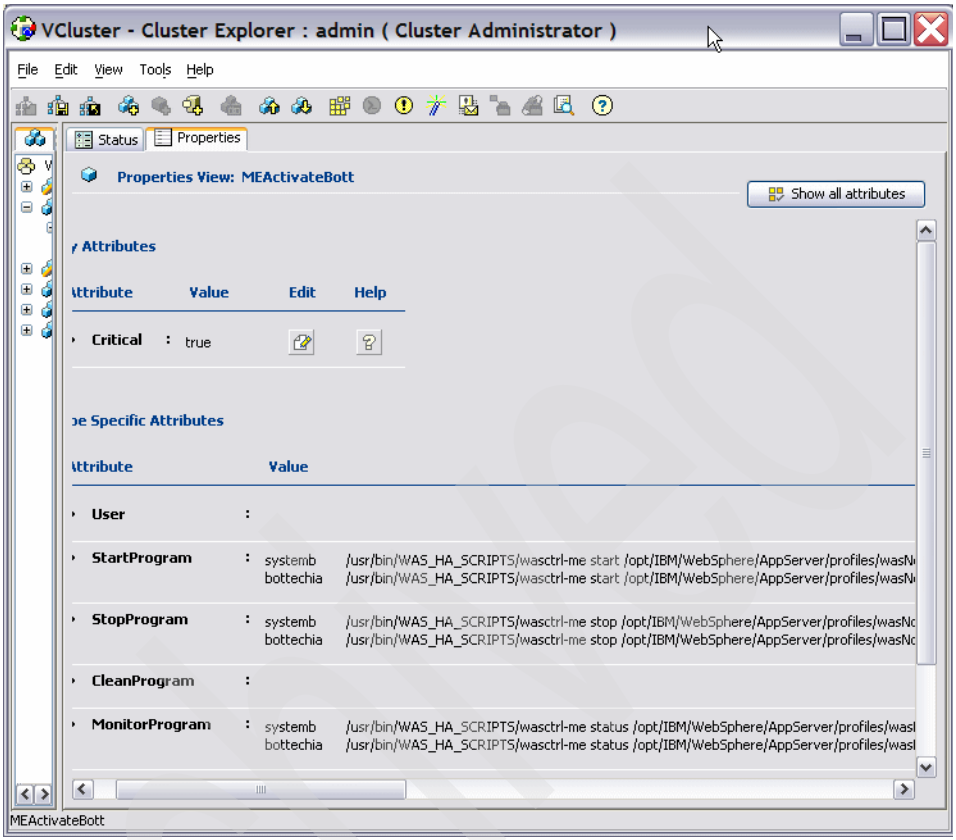


Figure 12-18 ME activate resource for ME NoOP policy

This resource specifies the script wasctrl-me as explained in “Scripts to start, stop, and monitor WebSphere resources” on page 331. It is important to understand the invocation parameters for the scripts on the different systems (primary and secondary). The script invocation for activating the ME on the *primary* system is:

```
/usr/bin/WAS_HA_SCRIPTS/wasctrl-me start
/opt/IBM/WebSphere/AppServer/profiles/wasNode02/ 49080 bottechia 48879
IBM_hc=TradeCluster,WSAF_SIB_BUS=TradeCluster,WSAF_SIB_MESSAGING_ENGINE=
TradeCluster.000-TradeCluster,type=WSAF_SIB /usr/bin/WAS_HA_SCRIPTS/
```

The first parameter is the action and indicates to start the ME. The second parameter is the install directory for the node. The next two parameters are the host and port for the servlet that is installed on the cluster member to perform the MBean actions. The next port is the **wsadmin** binding port. When activating against a cluster member that is not started (the primary cluster member), the scripts uses a JACL script and this is the SOAP port for running

wsadmin and the JACL script. The next parameter is the match criteria for selecting which ME to work against. The match criteria is the fully qualified name of the HA group associated with the messaging engine TradeCluster.000-TradeCluster. The last parameter is a pointer to where the scripts are installed.

Note: The wasctrl-me script has to be modified slightly. The script sets return codes for the status of the resource. VCS has defined ranges of return codes to indicate whether a resource status is online, offline or unknown. The script has to be modified so that the correct status value is returned for VCS. The script needs to have the following values modified:

- UNKNOWN=0
- ONLINE=110
- OFFLINE=100

The invocation for activating the ME recovery on the *secondary* system is:

```
/usr/bin/WAS_HA_SCRIPTS/wasctrl-me start
/opt/IBM/WebSphere/AppServer/profiles/wasNode01/ 49080 systemb 48879
IBM_hc=TradeCluster,WSAF_SIB_BUS=TradeCluster,WSAF_SIB_MESSAGING_ENGINE=
TradeCluster.000-TradeCluster,type=WSAF_SIB /usr/bin/WAS_HA_SCRIPTS/
```

Notice that the secondary invocation indicates to run on the secondary system (SystemB) but to activate the ME that failed on the primary system (Bottechia). The parameters for the node install directory, the servlet port, host name, and SOAP port are all pointing to the secondary system (even though some of the values are the same as the primary system). The parameter for the match criteria indicates the ME associated with the primary cluster member (Bottechia).

ME activate for system SystemB

This group is very similar to the previous group. This group is configured as a failover group, with the primary system being SystemB and the secondary system being Bottechia. In our example, we configured this group with one application resource.

- Activate ME application resource

  This resource activates, deactivates and monitors the messaging engine. For the primary system (SystemB), this resource activates the messaging engine. In a failure situation, the messaging engine on the cluster member on the secondary system (Bottechia) will be activated.

  This resource specifies the script wasctrl-me as explained in “Scripts to start, stop, and monitor WebSphere resources” on page 357. It is important to understand the invocation parameters for the scripts on the different systems

(primary and secondary). The script invocation for activating the ME on the *primary* system is:

```
/usr/bin/WAS_HA_SCRIPTS/wasctrl-me start
/opt/IBM/WebSphere/AppServer/profiles/wasNode01/ 9080 systemb 8879
IBM_hc=TradeCluster,WSAF_SIB_BUS=TradeCluster,WSAF_SIB_MESSAGING_ENGINE=
TradeCluster.001-TradeCluster,type=WSAF_SIB /
```

The first parameter is the action and indicates to start the ME. The second parameter is the install directory for the node. The next two parameters are the host and port for the servlet that is installed on the cluster member to perform the MBean actions. The next port is the **wsadmin** binding port. When activating against a cluster member that is not started (the primary cluster member), the script uses a JACL script and this is the SOAP port for running **wsadmin** and the JACL script. The next parameter is the match criteria for selecting which ME to work against. The match criteria is the fully qualified name of the HA group associated with the messaging engine TradeCluster.001-TradeCluster. The last parameter is a pointer to where the scripts are installed.

The invocation for activating the ME recovery on the *secondary* system is:

```
/usr/bin/WAS_HA_SCRIPTS/wasctrl-me start
/opt/IBM/WebSphere/AppServer/profiles/wasNode02/ 49080 bottechia 48879
IBM_hc=TradeCluster,WSAF_SIB_BUS=TradeCluster,WSAF_SIB_MESSAGING_ENGINE=
TradeCluster.001-TradeCluster,type=WSAF_SIB /usr/bin/WAS_HA_SCRIPTS/
```

Notice that the secondary invocation indicates to run on the secondary system (Bottechia) but to activate the ME that failed on the primary system (SystemB). The parameters for the node install directory, the servlet port, host name, and SOAP port are all pointing to the secondary system (even though some of the values are the same as the primary system). The parameter for the match criteria indicates the ME associated with the primary cluster member (SystemB).

12.6.3 Testing messaging engine with NoOP policy failover

When all resources are set up and running on the systems we recommend you test the setup to see that everything is working as expected. Possible tests include:

- Shut down the system where the application server is running.
- Kill the application server process using the **kill -9** command.

When a messaging engine starts in a server, messages are written to the SystemOut.log file. Therefore, monitor this file on the failed-over system for messages similar to the ones shown in Example 12-1 on page 480.

Example 12-1 Message engine start messages

```
[8/25/05 11:18:03:743 CDT] 00000045 SibMessage    I
[TradeCluster:TradeCluster.000-TradeCluster] CWSIS1538I: The messaging engine,
ME_UUID=02BF0BF501CE0540, INC_UUID=373ad6a9ee6e8153, is attempting to obtain an
exclusive lock on the data store.
...
[8/25/05 11:18:04:037 CDT] 00000045 SibMessage    I
[TradeCluster:TradeCluster.000-TradeCluster] CWSIS1537I: The messaging engine,
ME_UUID=02BF0BF501CE0540, INC_UUID=373ad6a9ee6e8153, has acquired an exclusive
lock on the data store.
...
[8/25/05 11:18:11:227 CDT] 00000027 SibMessage    I
[TradeCluster:TradeCluster.000-TradeCluster] CWSID0016I: Messaging engine
TradeCluster.000-TradeCluster is in state Started.
```

Alternatively, you can open the Administrative Console and go to **Core groups** → **Core group settings** → ***CoreGroupName*** (most probably DefaultCoreGroup) → Runtime tab.

Enter `WSAF_SIB_MESSAGING_ENGINE=`*`Your_ME`* into the Group name properties field, and click **Show groups (**for example, `WSAF_SIB_MESSAGING_ENGINE=TradeCluster.000-TradeCluster`). You could also leave the asterisk (*) in the Group name properties field, and click **Show groups** to display all High availability groups.

Click the link for your HA group to display the Name of the Server the ME runs on. This panel also shows the status of the ME. See Figure 12-19. Verify this panel before and after the failover.

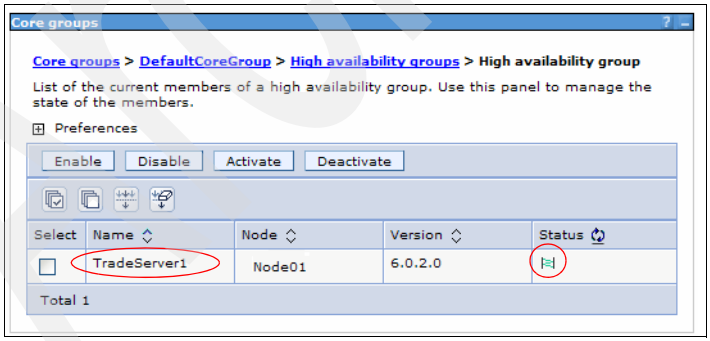


Figure 12-19 ME status and server it runs on

12.7 Reference

- VERITAS Web site

  http://www.veritas.com

- *VERITAS Cluster Server Agent 3.5 for WebSphere MQ Installation and Configuration Guide*

  http://ftp.support.veritas.com/pub/support/products/ClusterServer_UNIX/273084.pdf

- *VERITAS Cluster Server Enterprise Agent 4.0 for DB2 Installation and Configuration Guide*

  http://ftp.support.veritas.com/pub/support/products/ClusterServer_UNIX/270374.pdf

13

WebSphere and Sun Cluster

This chapter describes setting up WebSphere Application Server to run on a Sun Cluster environment. We discuss how to install WebSphere Application Server Network Deployment and how to create a Sun Cluster resource to restart WebSphere processes.

We are using the IBM Trade Performance Benchmark Sample for WebSphere Application Server application for our scenario. We refer to this application as Trade 6 throughout the chapter.

Important: This chapter contains information and material from Sun Microsystems, Inc. This information was printed by permission of Sun Microsystems, Inc.

13.1 Introduction to Sun Cluster

The Sun Cluster, or SunPlex™ system, is an integrated hardware and software solution that is used to create highly available and scalable services. The Sun Cluster system extends the Solaris operating environment into a cluster operating system. There are existing resource types for DB2 and WebSphere MQ. Resource types can also be made for various WebSphere Application Server processes.

13.1.1 How Sun Cluster works

The Sun Cluster system achieves high availability through a combination of hardware and software. The redundant cluster interconnects storage and public networks to protect against single points of failure. The cluster software continuously monitors the health of member nodes and prevents failing nodes from participating in the cluster, protecting against data corruption. Also, the cluster monitors services and their dependent system resources, and fails over or restarts services in case of failures. Sun Cluster supports up to 16 nodes (in Sun Cluster 3.1).

Sun Cluster uses *agents* to simplify cluster configuration. Each type of resource supported in a cluster is associated with an agent. An agent is an installed program designed to control a particular resource type. For Sun Cluster to bring a certain resource online, it does not need to understand it, but simply pass the online command to the agent. Sun Cluster has Oracle, DB2, Informix®, and SyBase agents. Many other enterprise agents are also available. Some of these agents come with the base cluster software, and some require additional purchase. Sun Cluster can use the Solaris Volume Manager or the VERITAS Volume Manager.

The *Resource Group manager* handles high availability and scalability in Sun Cluster. It manages the resource types (also known as data services), resource groups and resources.

Sun Cluster supports both Active/Passive failover and Active/Active availability. Sun Cluster uses the terms *failover* and *scalable* to describe these activities.

Sun Cluster can:

- Reduce or eliminate system downtime because of software or hardware failures.
- Ensure availability of data and applications to users, regardless of the kind of failure that would normally take down a single-server system.

- Increase application throughput by enabling services to scale to additional processors by adding nodes to the cluster.
- Provide enhanced availability of the system by enabling you to perform maintenance without shutting down the entire cluster.

Sun Cluster configurations tolerate the following types of single-point failures:

- Server operating environment failure because of a crash or a panic
- Data service failure
- Server hardware failure
- Network interface failure
- Disk media failure

As depicted in Figure 13-1, clients access the primary system.

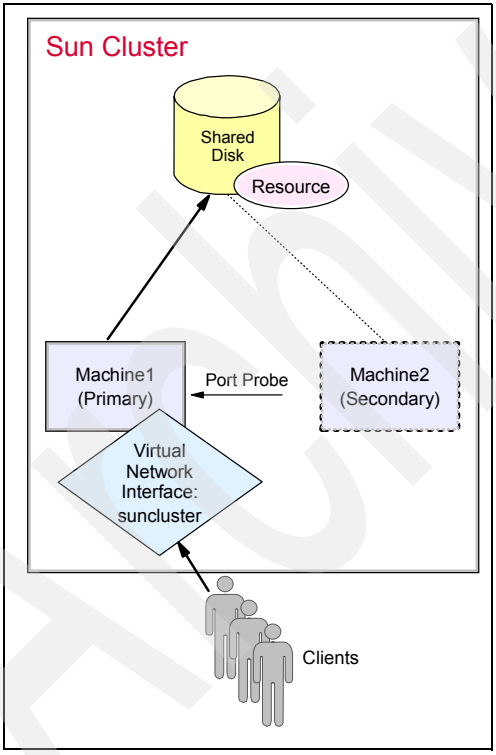


Figure 13-1 Sun Cluster setup: running on primary machine

In the case of a failure, the clients failover to the second machine as shown in Figure 13-2.

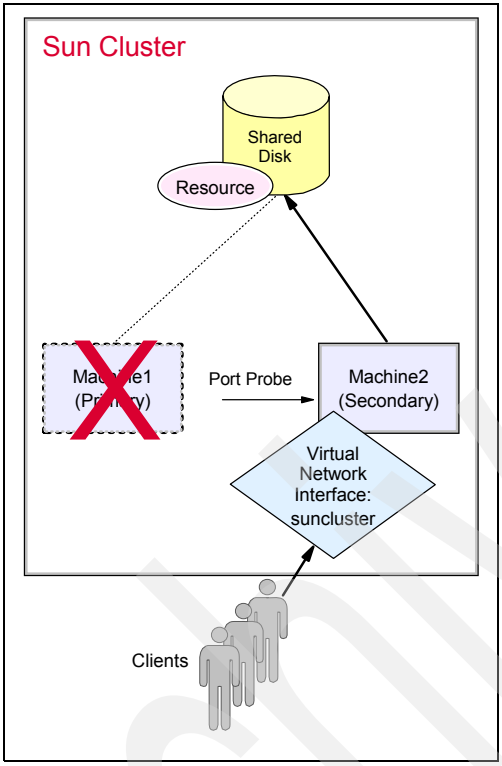


Figure 13-2 Sun Cluster setup: primary machine fails, failover to secondary machine

You can find more information about Sun Cluster software at:

http://www.sun.com/clusters

Notes:

- The current version of Sun Cluster is 3.1 8/05.
- We used Sun Cluster 3.0 during our tests.

13.1.2 Configuration basics of Sun Cluster

There are several steps to configure a Sun Cluster for WebSphere:

1. Connect and configure the shared disk subsystem.

   Files are stored in a mirrored or RAID shared disk array that can be connected physically to two nodes (multihost disk). When one node fails, the

other node can access the same data and log files. You can configure the shared disk using either Solaris Volume Manager or VERITAS Volume Manager.

2. Install and configure the Sun Cluster data service software. Refer to the appropriate Sun Cluster Software Installation and the Sun Cluster Software Administration guides for your Sun Cluster version, which is available at:

   http://docs.sun.com/app/docs

3. Start the Sun Cluster and take ownership of the disk group using this command:

   ```
   scadmin startcluster
   ```

4. Create a resource group using the **scrgadm** command. See the article *How to Create a Failover Resource Group*, which is available at:

   http://docs.sun.com/app/docs/doc/817-1526/6mh8j0jd4?a=view#babhbcji

5. Add logical host name resources and shared address resources. See the article *Adding Resources to Resource Groups*, which is available at:

   http://docs.sun.com/app/docs/doc/817-1526/6mh8j0jd6?a=view

Using this basic set up, WebSphere Application Server resources can be added to the resource group.

13.1.3 Managing resources

Resource groups and resources can be managed with the **scswitch** command. The resource group can be stopped, started and switched between systems. Resources can be enabled and disabled. Enabling a resource kicks off the start script and Sun Cluster begins to monitor it. Disabling a resource initiates the stop script.

To switch the resource group to another system, use the following command:

```
scswitch -Z -g resource_group_name
```

To enable a resource and initiate the start script, use the following command:

```
scswitch -e -j resource_name
```

To disable a resource and initiate the stop script, use the following command:

```
scswitch -n -j resource_name
```

Refer to the man page for **scswitch** from Sun at:

http://docs.sun.com/app/docs/doc/816-5251/6mbdimpp2?q=scswitch&a=view

To review the status of the resource groups and resources, use the **scstat** command. Example 13-1 shows output for the **scstat** command.

Example 13-1 Result of scstat

```
------------------------------------------------------------------

-- Cluster Nodes --

                    Node name           Status
                    ---------           ------
  Cluster node:     ws-cluster1         Online
  Cluster node:     ws-cluster2         Online

------------------------------------------------------------------

-- Cluster Transport Paths --

                    Endpoint            Endpoint            Status
                    --------            --------            ------
  Transport path:   ws-cluster1:qfe3    ws-cluster2:qfe3    Path online
  Transport path:   ws-cluster1:qfe0    ws-cluster2:qfe0    Path online

------------------------------------------------------------------

-- Quorum Summary --

  Quorum votes possible:      4
  Quorum votes needed:        3
  Quorum votes present:       4


-- Quorum Votes by Node --

                    Node Name           Present Possible Status
                    ---------           ------- -------- ------
  Node votes:       ws-cluster1         1        1       Online
  Node votes:       ws-cluster2         1        1       Online


-- Quorum Votes by Device --

                    Device Name         Present Possible Status
                    -----------         ------- -------- ------
  Device votes:     /dev/did/rdsk/d12s2 1        1       Online
  Device votes:     /dev/did/rdsk/d13s2 1        1       Online

------------------------------------------------------------------
```

```
-- Device Group Servers --

                         Device Group        Primary             Secondary
                         ------------        -------             ---------


-- Device Group Status --

                              Device Group        Status
                              ------------        ------

-----------------------------------------------------------------

-- Resource Groups and Resources --

            Group Name          Resources
            ----------          ---------
 Resources: db2_db2inst1_0-rg   suncluster db2_db2inst1_0-rs nodeagent-rs
wasmem09-rs wasmem10-rs
 Resources: cluster1TM          -
 Resources: cluster2TM          -


-- Resource Groups --

            Group Name          Node Name           State
            ----------          ---------           -----
     Group: db2_db2inst1_0-rg   ws-cluster1         Offline
     Group: db2_db2inst1_0-rg   ws-cluster2         Online

     Group: cluster1TM          ws-cluster1         Unmanaged
     Group: cluster1TM          ws-cluster2         Unmanaged

     Group: cluster2TM          ws-cluster2         Unmanaged
     Group: cluster2TM          ws-cluster1         Unmanaged


-- Resources --

            Resource Name       Node Name           State     Status Message
            -------------       ---------           -----     --------------
  Resource: suncluster          ws-cluster1         Offline   Offline
  Resource: suncluster          ws-cluster2         Online    Online -
LogicalHostname online.

  Resource: db2_db2inst1_0-rs   ws-cluster1         Offline   Offline
  Resource: db2_db2inst1_0-rs   ws-cluster2         Online    Online

  Resource: nodeagent-rs        ws-cluster1         Offline   Offline
```

```
Resource: nodeagent-rs        ws-cluster2         Online    Online

Resource: wasmem09-rs         ws-cluster1         Offline   Offline
Resource: wasmem09-rs         ws-cluster2         Online    Online

Resource: wasmem10-rs         ws-cluster1         Offline   Offline
Resource: wasmem10-rs         ws-cluster2         Online    Online
```

13.1.4 Using the Cluster Agent for WebSphere MQ

Sun Cluster has an existing data service or resource type for running WebSphere MQ. The agent is preprogrammed to start and to shut down, fault monitor, and perform automatic failover for the WebSphere MQ Integrator service.

For detailed instructions on using the MQ agent, see *Sun Cluster Data Service for WebSphere MQ Integrator Guide for Solaris OS*, which is available at:

http://docs.sun.com/app/docs/doc/817-4580

13.1.5 Using the Cluster Agent for DB2

Sun Cluster has an existing data service or resource type for running DB2. It can start, stop, monitor and administer DB2 UDB in a Sun Cluster 3.x environment.

For installation and configuration instructions for DB2 with Sun Cluster, see the IBM white paper, *IBM DB2 Universal Database and High Availability on Sun Cluster 3.x*, which is available at:

ftp://ftp.software.ibm.com/software/data/pubs/papers/suncluster.pdf

13.2 Planning and preparation

We assume that you have installed Sun Cluster on a group of Solaris systems. In the following sections, we describe how to configure WebSphere Application Server using a pair of Solaris machines with Sun Cluster installed. These machines share a disk and can access it at the same time. Both machines access the shared disk and a logical host name. In this example, a DB2 resource group and resource were previously created. We add a WebSphere Application Server resource to the existing resource group, db2_db2inst1_0-rg, which contains a host name resource and db2 resource.

This chapter discusses how to set up a Deployment Manager, Node Agent, and application servers in failover resources. We also provide information about setting up a Node Agent and application servers in a scalable environment.

The following are important names that we use in this example:

Solaris machine 1:	SHost01
Solaris machine 2:	SHost02
Shared disk:	/global/scudb
Logical host name:	suncluster.ibmredbook.com
Shared scalable host name:	a-suncluster.ibmredbook.com
Failover resource group:	db2_db2inst1_0-rg
Shared scalable resource group:	websphere_active-rg
Logical host name resource:	suncluster
Scalable host name resource:	a-suncluster

13.3 Deployment Manager

Making the Deployment Manager highly available using Sun Cluster requires that you install it on the shared disk. It is also possible to install WebSphere Network Deployment locally and only the profile on the shared disk. Nodes are installed on remote machines. We set up an Active/Passive configuration.

After installing the Deployment Manager, you need to create a Deployment Manager resource for Sun Cluster to start, stop, and monitor the Deployment Manager automatically. Then, you need to verify the scenario by failing over and monitoring the nodes and the restart of the Deployment Manager.

The end result should be a Deployment Manager that is available on two Sun systems, as pictured in Figure 13-3 on page 492.

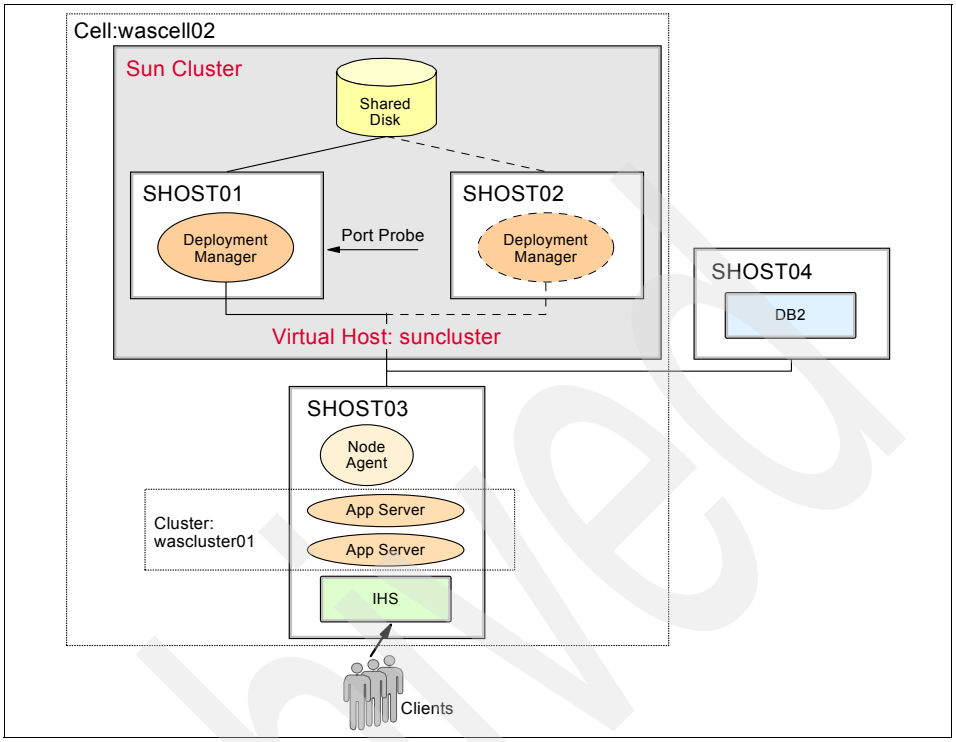


Figure 13-3 Deployment Manager configured on Sun Cluster (Active/Passive)

13.3.1 Installing WebSphere Network Deployment

To limit single points of failure, you should install the Deployment Manager on the Sun Cluster system and the nodes on separate systems. In our scenario, there is one Deployment Manager and one WebSphere Application Server node.

To install WebSphere:

1. Install WebSphere Network Deployment V6.0.1 onto the shared disk drive on SHost01.
2. Create a Deployment Manager profile. Provide the host name of the logical host name for the Sun Cluster setup. In our example, we use suncluster.ibmredbook.com.
3. Start the Deployment Manager, and verify that it starts successfully. Review the Deployment Manager logs and open the Administrative Console.
4. Stop the Deployment Manager.

The Deployment Manager is now ready for nodes to be federated on the Sun Cluster configuration.

13.3.2 Configuring Deployment Manager with Sun Cluster

In order for Sun Cluster to restart and monitor the Deployment Manager, you must create a resource for the Deployment Manager. This requires several steps using Solaris and Sun Cluster tools and commands, which are discussed in the sections that follow.

Creating a Deployment Manager package

Use the SunPlex Agent Builder to create a package that includes the start and stop commands for the Deployment Manager. To create the Deployment Manager package:

1. Open SunPlex Agent Builder:

   ```
   /usr/cluster/bin/scdsbuilder
   ```

2. In the Step 1 of the SunPlex Agent Builder window, fill in these fields:
   - Vendor Name: `IBM`
   - Application Name: `WAS`
   - Working Directory: `/workarea/agentbuilder`
   - Failover: selected
   - Resource Type: `ksh`

   Click **Create**.

3. Click **OK** in the success window.
4. Click **Next** in the SunPlex Agent Builder window.
5. In the Step 2 of the SunPlex Agent Builder window, you need to fill in the start and stop commands for the Deployment Manager in the appropriate fields.
   - For Start Command, enter the path to the **`startManager.sh`** command. In our example the path is:

     ```
     /global/scudb/WebSphere/ApplicationServer/profiles/itsoprofile/bin/
     startManager.sh
     ```

   - For Stop Command, enter the path to the **`stopManager.sh`** command. In our example the path is:

     ```
     /global/scudb/WebSphere/ApplicationServer/profiles/itsoprofile/bin/
     stopManager.sh
     ```

   - Leave the Probe Command field blank.

   Click **Configure**.

6. Click **OK** on the success box.

7. After the package has been created, click **Cancel** in the SunPlex Agent Builder window.
8. Open a command window and change to your Working Directory, then change to the IBMWAS/pkg directory. In our example you need to be in

   ```
   /workarea/agentbuilder/IBMWAS/pkg
   ```

9. Run the **pkgadd** command to add the newly created IBMWAS package:

   ```
   pkgadd -d . IBMWAS
   ```

10. Zip the agentbuilder directory and transfer it to the second system (SHost02). Unzip into the same directory structure, change to the /workarea/agentbuilder/IBMWAS/pkg directory, and run the **pkgadd** command from step 8 to create the package on this system as well.

Adding a Deployment Manager resource to Sun Cluster

Using the **scsetup** GUI and **scrgadm** command, add a Deployment Manager resource to Sun Cluster by following these steps:

1. On SHost01, run the **scsetup** GUI:

   ```
   /usr/cluster/bin/scsetup
   ```

2. On the Main Menu, select **2) Resource groups**.
3. On the Resource Group Menu, select **3) Add a data service resource to a resource group**.
4. On the Add a Data Service Resource to a Resource Group panel, enter `yes` to when asked if it is OK to continue. The following steps, 5 to 22 on page 496, are all performed on this panel.
5. Enter the existing resource group for the question when prompted for a new resource.

   To find the name of existing resource groups, run the **scstat** command in another window. Look for the Resource Groups and Resources section and the Group Name column. Sample output for our scenario is shown in Example 13-2.

Example 13-2 Existing resource groups and resources

```
-- Resource Groups and Resources --

            Group Name          Resources
            ----------          ---------
 Resources: db2_db2inst1_0-rg suncluster db2_db2inst1_0-rs
 Resources: cluster1TM          -
 Resources: cluster2TM          -
```

Here we find our Resource group called db2_db2inst1_0-rg.

6. Select **IBM.WAS WAS server for Sun Cluster** as the type of resource you want to add. This is the resource type that you created in “Creating a Deployment Manager package” on page 493.
7. Still on the same panel, answer `yes` when prompted whether the software for this service is installed on each node.
8. Answer `yes` when prompted to register this resource type now. The panel generates and runs the following command to add the resource type:

   ```
   scrgadm -a -t IBM.WAS
   ```

9. Press Enter to continue.
10. Enter a name for the resource. Our resource name is dmgr-rs.
11. When you see the section shown in Example 13-3, enter `yes` to override the default setting.

Example 13-3 Adding a Deployment Manager port for monitoring

```
This data service uses the "Port_list" property. The default
    "Port_list" for this data service is as follows:

        <NULL>

    Please check the documentation for this resource type for more
    information on how the list should be set for this resource.

    Do you want to override the default (yes/no) [no]?
```

12. Fill in a Deployment Manager port.

    The probe monitoring system from Sun Cluster tries to connect to this port to determine whether the Deployment Manager is running. We use the Deployment Manager’s SOAP port.

    To find the Deployment Manager’s SOAP port, open the WebSphere Administrative Console. Click **System administration** → **Deployment manager** → **Ports**. Open **SOAP_CONNECTOR_ADDRESS**. The SOAP port number for our Deployment Manager is 8879, thus, for Port number (Ctrl+D to finish), enter 8879.
13. Enter `yes` to the question whether this is a TCP port.
14. Press Ctrl+D to stop entering ports.
15. The panel now confirms the TCP port just entered, answer `yes` if it is correct.
16. Press Enter to continue.
17. When prompted for extension properties that you would like to set, enter `no`. However, if you know that you do want to change properties, you can enter

yes and change properties. The list of properties might look similar to that shown in Example 13-4.

Example 13-4 Extension properties that can be updated

```
Here are the extension properties for this resource:

        Property Name             Default Setting
        =============             ===============

        Confdir_list              <NULL>
        Monitor_retry_count       4
        Monitor_retry_interval    2
        Probe_timeout             30
        Child_mon_level           -1

    Please enter the list of properties you want to set:
      (Type Ctrl-D to finish  OR  "?" for help)
```

18. Answer yes when prompted to proceed with the update if you are ready to add the Deployment Manager resource.

19. Based on your entries, the Add a Data Service Resource to a Resource Group panel generates and runs this command:

    ```
    scrgadm -a -j dmgr-rs -g db2_db2inst1_0-rg -t IBM.WAS -y Scalable=false
    -y Port_list=8879/tcp
    ```

    If something goes wrong with adding the resource, you can copy and reuse this command instead of stepping through **scsetup** again.

20. Press Enter to continue.

21. Answer no when prompted whether you want to enable this resource.

22. Enter q twice to exit **scsetup**.

23. Update the Thorough_Probe_Interval and Retry_Interval properties.

    Updating these properties allows the Deployment Manager enough time to start before Sun Cluster checks to see if it is running. If Sun Cluster probes too soon, it tries to stop and restart the Deployment Manager in the middle of its initial start process. The Retry_Interval must be equal to or greater than the Thorough_Probe_Interval multiplied by the Retry_Count. To update these properties:

    a. Update the Retry_Interval to 1000:

       ```
       scrgadm -c -j dmgr-rs -y Retry_Interval=1000
       ```

    b. Update the Thorough_Probe_Interval:

       ```
       scrgadm -c -j dmgr-rs -y Thorough_Probe_Interval=500
       ```

These values assume the Retry_Count is set to 2.

24. In a command window, activate the resource using the **scswitch** command:

```
scswitch -e -j dmgr-rs
```

Review the Deployment Manager's log for a successful start. If there are problems starting, you can stop the resource using the **-n** parameter on **scswitch**:

```
scswtich -n -j dmgr-rs
```

25. Use the **scstat** command to review the status of the resources. The dmgr-rs resource should be added to the previously selected resource group listed in the Resource Groups and Resources section. See Example 13-5.

Example 13-5 Resources including the dmgr-rs resource

```
-- Resource Groups and Resources --

            Group Name          Resources
            ----------          ---------
 Resources: db2_db2inst1_0-rg suncluster db2_db2inst1_0-rs dmgr-rs
 Resources: cluster1TM          -
 Resources: cluster2TM          -
```

Sun Cluster can now control the Deployment Manager. Sun Cluster is able to stop and start the Deployment Manager and checks the SOAP port for its status.

13.3.3 Completing the WebSphere cell

A few more steps are required to complete the cell's configuration, such as federating nodes, possible enabling security, and installing the application.

Federating nodes

If you do not have a node already federated, configure a custom profile on a remote system and federate it to the Deployment Manager on the Sun Cluster setup. The host name for the Deployment Manager is the logical host name. For our configuration, the **addNode** command looks as follows:

```
addNode.sh suncluster 8879
```

Security considerations

Enable security if necessary. Use the soap.client.props properties file to enter the user and password for stopping the Deployment Manager. If you do not add the user name and password to the soap.client.properties file, the **stopManager** command expects the properties on the command line when security is enabled. This fails because the parameters are not part of the Sun Cluster stop script.

To update and encode soap.client.props:

1. Go to the Deployment Manager's profile directory. Under the properties directory, open soap.client.props.
2. Add the user name and password to the JMX SOAP connector identity section:
   - com.ibm.SOAP.loginUserid=*username*
   - com.ibm.SOAP.loginPassword=*password*
3. Use the PropFilePasswordEncoder tool to encode the login password. Pass com.ibm.SOAP.loginPassword as the property to encode. From the Deployment Manager's profile bin directory, run:

   ```
   ./PropFilePasswordEncoder.sh ../properties/soap.client.props
   com.ibm.SOAP.loginPassword
   ```

   After encoding, the properties file might be rearranged. A backup properties file is also created under the name of soap.client.props.bak.

You can find more information about this topic in the InfoCenter article *Protecting plain text passwords*.

Installing Trade 6

You need to install Trade 6 on the Deployment Manager. You can download Trade 6 from:

```
http://www.ibm.com/software/webservers/appserv/was/performance.html
```

Follow the directions included in the Trade 6 download package for setup and installation, or see Chapter 8 of *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392 for instructions on how to install and configure Trade 6.

If you use the trade.jacl script that is provided with the package to configure and install the application, you might need to include the **-conntype** and **-host** parameters if **wsadmin** cannot find the local host:

```
/global/scudb/WebSphere/AppServer/bin/wsadmin.sh -conntype soap -host
suncluster -f trade.jacl
```

If **wsadmin** cannot find the local host, you see the message that is shown in Example 13-6.

Example 13-6 wsadmin fails to find local host

```
WASX7023E: Error creating "SOAP" connection to host "localhost"; exception
information:
com.ibm.websphere.management.exception.ConnectorNotAvailableException:
com.ibm.websphere.management.exception.ConnectorNotAvailableException: Failed
to get a connection with IP address associated with hostname localhost
```

Start the Trade application and verify that it works correctly by populating and trading.

13.3.4 Testing Deployment Manager failover

After configuring the Deployment Manager and Sun Cluster, verify that the Deployment Manager will failover. The application should continue to run uninterrupted and the Node Agent should reconnect to the Deployment Manager after restart.

Two possible tests include:

- Killing the server process and ensuring that Sun Cluster restarts the Deployment Manager.
- Taking down a machine and verifying that the Deployment Manager remains active or restarts on the second machine.

Failing the Deployment Manager

To verify that Sun Cluster can monitor and restart the Deployment Manager, you can kill the Deployment Manager process:

1. On SHost01, use **scstat** to check where the resource group is located. If it is active on SHost02, use the **scswitch** command to swap to SHost01:

   ```
   scswitch -Z -g db2_db2inst1_0-rg
   ```

2. Verify that the Deployment Manager is started. Either open the Administrative Console or review the log for the following message:

   ```
   WSVR0001I: Server dmgr open for e-business.
   ```

   If the Deployment Manager is not running, start it using the **scswitch** command:

   ```
   scswitch -e -j dmgr-rs
   ```

3. Check the log for the Node Agent. It should successfully synchronize with the Deployment Manager as indicated by the following message:

   ```
   ADMSO003I: The configuration synchronization completed successfully.
   ```

4. Find the process number of the Deployment Manager by looking at the .pid file in the Deployment Manager's log directory. Use the **kill** command to end the process:

   ```
   kill -9 pid
   ```

5. Continue trading with the Trade 6 application. The application should not be affected.

6. Review the Node Agent's log. It notes that the Deployment Manager is no longer available as shown in Example 13-7.

Example 13-7 Node Agent log, Deployment Manager not available

```
...DCSV1113W: DCS Stack DefaultCoreGroup at Member wascell02\wasna01\nodeagent:
Suspected another member because the outgoing connection to the other member
was closed. Suspected member is wascell02\wasdmgr02\dmgr. DCS logical channel
is Connected|Ptp.
...DCSV8053I: DCS Stack DefaultCoreGroup at Member wascell02\wasna01\nodeagent:
View change in process. Excluded members are [wascell02\wasdmgr02\dmgr].
```

The Node Agent also fails to synchronize with the Deployment Manager, as shown in Example 13-8.

Example 13-8 Node Agent fails to synchronize

```
...ADMSO015E: The synchronization request cannot be completed because the node
agent cannot communicate with the deployment manager.
...ADMSO016I: The configuration synchronization failed.
```

7. Watch the Deployment Manager log for restarting. Review the log for the following message that indicates a successful start:

   ```
   WSVR0001I: Server dmgr open for e-business.
   ```

8. When the Deployment Manager restarts, return to the Node Agent log. It should recognize that the Deployment Manager restarted, discover it, and synchronize successfully, as shown in Example 13-9.

Example 13-9 Node Agent discovers that the Deployment Manager restarted.

```
...DCSV1032I: DCS Stack DefaultCoreGroup at Member wascell02\wasna01\nodeagent:
Connected a defined member wascell02\wasdmgr02\dmgr.
...ADMD0023I: The system discovered process (name: dmgr, type:
DeploymentManager, pid: 12278)
...NodeSyncTask  A   ADMSO003I: The configuration synchronization completed
successfully.
```

This example shows the Deployment Manager process dying and being managed by Sun Cluster.

Rebooting a Sun Cluster machine

To test that the Deployment Manager is highly available on both machines using a failover configuration with Sun Cluster, reboot the machine acting as the primary.

1. On SHost01, use **scstat** to check where the resource group is located. If it is active on SHost02, use the **scswitch** command to swap to SHost01:

   ```
   scswitch -Z -g db2_db2inst1_0-rg
   ```

2. Verify that the Deployment Manager is started. Either open the Administrative Console or review the log for the following message:

   ```
   WSVR0001I: Server dmgr open for e-business.
   ```

   If the Deployment Manager is not running, start it using the **scswitch** command:

   ```
   scswitch -e -j dmgr-rs
   ```

3. Check the Node Agent log. It should synchronize successfully with the Deployment Manager, as indicated by the following message:

   ```
   ADMS0003I: The configuration synchronization completed successfully.
   ```

4. Reboot SHost01.
5. Ping the logical host name, suncluster.ibmredbook.com. It should be available shortly.
6. On SHost02, review the Deployment Manager log for the Deployment Manager restart.
7. Review the Node Agent log, it should synchronize successfully with the Deployment Manager, as indicated by the following message:

   ```
   ADMS0003I: The configuration synchronization completed successfully.
   ```

8. During the failover, continue trading with Trade 6. It should behave normally.

13.4 Node Agent and application servers

Using Sun Cluster for WebSphere Application Server nodes might require making both the Node Agent and application servers highly available. We are using an Active/Passive configuration. In our example, a WebSphere node is installed on the shared disk. Both machines have access to the same files. It is also possible to install the Network Deployment binaries locally and only the node profile on the shared disk.

We assume that the Deployment Manager is installed, configured, and started on a remote machine. After installing the node, create resources for Sun Cluster to automatically start, stop, and monitor the Node Agent and application servers. Then, verify the scenario by failing over and monitoring the Node Agent and the restart of the servers.

The end result should be a Node Agent and applications servers that are available on two Sun systems, as pictured in Figure 13-4.

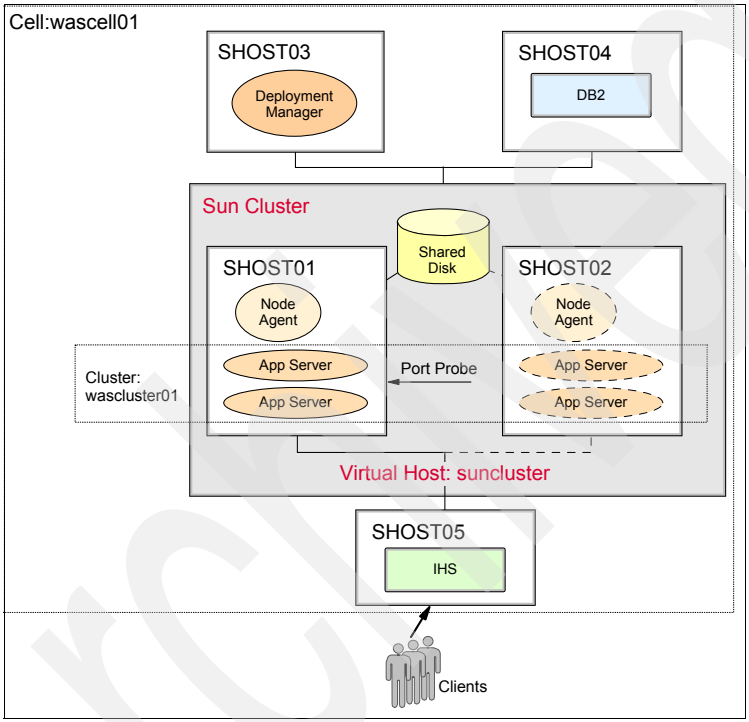


Figure 13-4 Node Agent and application servers configured with Sun Cluster

13.4.1 Installing a Node Agent and application server

To install a Node Agent and application server (or servers):

1. Install WebSphere Network Deployment V6.0.1 onto the shared disk drive on SHost01.
2. Create a custom profile. Enter the logical host name of the Sun Cluster setup in the Hostname field. For our scenario, we use suncluster.ibmredbook.com.
3. Federate the node to the Deployment Manager on the remote machine.

13.4.2 Completing the configuration

To finish the configuration, you need to configure a Web server, create a cluster and cluster members, install the sample application, change settings for the application servers, and enable security if desired.

Configuring a Web server

Install a Web server. In this example, we installed IBM HTTP Server and configured it in the cell using the name http1. Refer to Chapter 6 of *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392 for information about how to configure a Web server in a cell.

Installing Trade 6

Install Trade 6 on the Deployment Manager. Follow the directions that are included in the Trade 6 download package for setup and installation, and use the trade.jacl script for the setup. For additional information about how to install and configure Trade 6, refer to Chapter 8 of *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392.

During the installation, create an application server cluster with two cluster members. In our example, we used the following names:

- Cluster name: `wascluster01`
- Cluster members: `wasmember09, wasmember10`

Map the Trade 6 Web modules to the previously created Web server, http1.

Changing application server monitoring policies

Change the automatic restart property of the application servers. This allows Sun Cluster to gain control over restarting the servers and prevents the Node Agent and Sun Cluster from trying to restart an application server at the same time.

To disable the automatic restart for the application servers:

1. In the Administrative Console, go to **Servers** → **Application servers** → **wasmember09** → **Java and Process Management** → **Process Definition** → **Monitoring Policy**.
2. Deselect the Automatic restart field. Click **OK** and save your changes.
3. Repeat this for wasmember10.
4. Restart the application servers and the Node Agent.

You can find more information about this option in the InfoCenter article *Monitoring policy settings*.

Security considerations

Enable security if necessary. Use the soap.client.props properties file on the node to enter the user name and password for stopping the Node Agent and application servers. If you do not add the user name and password to the soap.client.properties file, the **stopNode** and **stopServer** commands expect the properties on the command line, which does not appear as part of the Sun Cluster start and stop scripts.

Encode the soap.client.props property file:

1. Go to the node's profile directory. Under the properties directory, open soap.client.props.
2. Add the user name and password to the JMX SOAP connector identity section:
   - com.ibm.SOAP.loginUserid=*username*
   - com.ibm.SOAP.loginPassword=*password*
3. Use the PropFilePasswordEncoder tool to encode the login password. Pass com.ibm.SOAP.loginPassword as the property to encode. From the node's profile bin directory, run:

   ```
   ./PropFilePasswordEncoder.sh ../properties/soap.client.props
   com.ibm.SOAP.loginPassword
   ```

   After encoding, the properties file might be rearranged. A backup properties file is also created under the name of soap.client.props.bak.

Also, see the InfoCenter article *Protecting plain text passwords*.

Testing the application

Start the Trade 6 application and verify that it works correctly by populating the Trade 6 database and trading. Stop the Node Agent and cluster.

13.4.3 Configuring Sun Cluster to run the Node Agent

To enable Sun Cluster to restart the Node Agent, create a new resource type and resource. In our example, an existing resource group is used.

Creating a Node Agent package

Use the SunPlex Agent Builder to create a package with start and stop commands for the Node Agent. To create a Node Agent package:

1. Open SunPlex Agent Builder:

   ```
   /usr/cluster/bin/scdsbuilder
   ```

2. In the Step 1 of the SunPlex Agent Builder window, fill in these fields:
   - Vendor Name: `IBM`
   - Application Name: `WASNA`
   - Working Directory: `/workarea/agentbuilder2`
   - Failover: selected
   - Resource Type: *ksh*

   Click **Create**.
3. Click **OK** in the success window.
4. Click **Next** on the SunPlex Agent Builder window.
5. In the Step 2 of the SunPlex Agent Builder window, you need to fill in the start and stop commands for the Node Agent in the appropriate fields.

   For Start Command, enter the path to the **startNode.sh** command. In our example the path is:

   ```
   /global/scudb/WebSphere/ApplicationServer/profiles/itsoprofile3/bin/
   startNode.sh
   ```

   For Stop Command, enter the path to the **stopNode.sh** command. In our example the path is:

   ```
   /global/scudb/WebSphere/ApplicationServer/profiles/itsoprofile3/bin/
   stopNode.sh
   ```

   Leave the Probe Command field blank.

   Click **Configure**.
6. Click **OK** on the success box.
7. After creation of the package, click **Cancel** on the SunPlex Agent Builder window.
8. Open a command window and change to your Working Directory, then change to the IBMWASNA/pkg directory. In our example you need to be in:

   /workarea/agentbuilder2/IBMWASNA/pkg

   Run the **pkgadd** command to add the newly created IBMWASNA package:

   ```
   pkgadd -d . IBMWASNA
   ```
9. Zip the agentbuilder2 directory and transfer it to the second system (SHost02). Unzip into the same directory structure, change to the IBMWASNA/pkg directory, and run the **pkgadd** command from step 8 to create the package on this system as well:

   ```
   pkgadd -d . IBMWASNA
   ```

Add Node Agent resource to Sun Cluster

Using the **scsetup** GUI and **scrgadm** command, add a Node Agent resource to Sun Cluster:

1. On SHost01, run the **scsetup** GUI:

   ```
   /usr/cluster/bin/scsetup
   ```

2. On the Main Menu, select **2) Resource groups**.
3. On the Resource Group Menu, select **3) Add a data service resource to a resource group**.
4. On the Add a Data Service Resource to a Resource Group panel, enter `yes` when prompted on wether to continue. The following steps, 5 to 22 on page 508, are all performed on this same panel.
5. Enter the existing resource group when prompted for the group that you want to add the new resource. To find the name of existing resource groups, run the **scstat** command in another window. Look for the Resource Groups and Resources section and the Group Name column. Sample output for our scenario is shown in Example 13-10.

Example 13-10 Existing resource groups and resources

```
-- Resource Groups and Resources --

            Group Name          Resources
            ----------          ---------
 Resources: db2_db2inst1_0-rg suncluster db2_db2inst1_0-rs
 Resources: cluster1TM          -
 Resources: cluster2TM          -
```

   Here we find our Resource group called db2_db2inst1_0-rg.

6. Select **IBM.WAS WASNA server for Sun Cluster** when prompted for the type of resource that you want to add. This is the resource type created in "Creating a Node Agent package" on page 504.
7. Still on the same panel, answer `yes` to the question on whether the software for this service is installed on each node.
8. Answer `yes` to the question of whether it is okay to register this resource type now. The panel generates and runs the following command to add the resource type:

   ```
   scrgadm -a -t IBM.WASNA
   ```

9. Press Enter to continue.
10. When prompted for the name of the resource that you want to add, enter a name for the resource. Our resource name is `nodeagent-rs`.

11. When you see the section shown in Example 13-11, enter `yes` to override the default setting.

Example 13-11 Adding a Deployment Manager port for monitoring

```
This data service uses the "Port_list" property. The default
    "Port_list" for this data service is as follows:

        <NULL>

    Please check the documentation for this resource type for more
    information on how the list should be set for this resource.

    Do you want to override the default (yes/no) [no]?
```

12. Fill in a Node Agent port. The probe monitoring system from Sun Cluster tries to connect to this port to determine whether the Node Agent is running. We use the Node Agent's SOAP port.

    To find the Node Agent's SOAP port, open the WebSphere Administrative Console. Select **System administration** → **Node agents** → **nodeagent** → **Ports**. Open **SOAP_CONNECTOR_ADDRESS**.

    The SOAP port number for our Node Agent is 8880, thus, for Port number (Ctrl+D to finish), enter 8880.

13. Enter `yes` to the question whether this is a TCP port.

14. Press Ctrl+D to stop entering ports.

15. The panel now confirms the TCP port just entered, answer `yes` if it is correct.

16. Press Enter to continue.

17. When prompted to set extension properties, enter `no`. However, if you know that you do want to change properties, you can enter `yes` and change properties. The list of properties might look similar to that shown in Example 13-12.

Example 13-12 Extension properties that can be updated

```
Here are the extension properties for this resource:
         Property Name             Default Setting
         =============             ===============
       Confdir_list              <NULL>
        Monitor_retry_count       4
        Monitor_retry_interval    2
        Probe_timeout             30
        Child_mon_level           -1
      Please enter the list of properties you want to set:
      (Type Ctrl-D to finish  OR  "?" for help)
```

18. Answer `yes` when prompted on whether to proceed with the update, if you are ready to add the Node Agent resource.

19. Based on your entries, the Add a Data Service Resource to a Resource Group panel generates and runs this command:

```
scrgadm -a -j nodeagent-rs -g db2_db2inst1_0-rg -t IBM.WASNA -y
Scalable=false -y Port_list=8880/tcp
```

    If something goes wrong with adding the resource, you can copy and reuse this command instead of stepping through **scsetup** again.

20. Press Enter to continue.

21. Answer `no` when prompted on whether you want to enable this resource.

22. Enter `q` twice to exit **scsetup**.

23. Update the Thorough_Probe_Interval and Retry_Interval properties.

    Updating these properties allows the Node Agent enough time to start before Sun Cluster checks to see if it is running. If Sun Cluster probes too soon, it tries to stop and restart the Node Agent in the middle of its initial start process. The Retry_Interval must be equal to or greater than the Thorough_Probe_Interval multiplied by the Retry_Count. To update these properties:

    a. Update the Retry_Interval to 1000:

```
scrgadm -c -j nodeagent-rs -y Retry_Interval=1000
```

    b. Update the Thorough_Probe_Interval:

```
scrgadm -c -j nodeagent-rs -y Thorough_Probe_Interval=500
```

    These values assume the Retry_Count is set to 2.

24. In a command window, activate the resource using the **scswitch** command:

```
scswitch -e -j nodeagent-rs
```

    Review the Node Agent's log for a successful start. If there are problems starting, you can stop the resource using the -n parameter on **scswitch**:

```
scswtich -n -j nodeagent-rs
```

25. Use **scstat** to review the status of the resources. The nodeagent-rs resource should be added to the previously selected resource group under the Resource Groups and Resources heading as shown in Example 13-13.

Example 13-13 Resources including the nodeagent-rs resource

```
-- Resource Groups and Resources --

            Group Name          Resources
            ----------          ---------
 Resources: db2_db2inst1_0-rg suncluster db2_db2inst1_0-rs nodeagent-rs
```

```
Resources: cluster1TM          -
Resources: cluster2TM          -
```

Sun Cluster can now control the Node Agent. Sun Cluster is able to stop and start the Node Agent and checks the SOAP port for its status.

13.4.4 Configure Sun Cluster to run application server

To allow Sun Cluster to restart the application server, create a new resource type and resource. We use an existing resource group in our scenario.

Creating an application server package

Use the SunPlex Agent Builder to create a package with start and stop commands for an application server. To create an application server package:

1. Open SunPlex Agent Builder:

   ```
   /usr/cluster/bin/scdsbuilder
   ```

2. In the Step 1 of the SunPlex Agent Builder window, fill in these fields:
   - Vendor Name: `IBM`
   - Application Name: `WASA9`
   - Working Directory: `/workarea/agentbuilder3`
   - Failover: selected
   - Resource Type: *`ksh`*

   Click **Create**.

3. Click **OK** on the success window.
4. Click **Next** on the SunPlex Agent Builder window.
5. In the Step 2 of the SunPlex Agent Builder window, you need to fill in the start and stop commands for the application server in the appropriate fields.

   For Start Command, enter the path to the **`startServer.sh`** command. In our scenario, the path is as follows:

   ```
   /global/scudb/WebSphere/ApplicationServer/profiles/itsoprofile3/bin/
   startServer.sh
   ```

   After selecting the command, add the name of the application server to be started at the end of the Start Command field. In this example, we add wasmember09 to the end of the command:

   ```
   /global/scudb/WebSphere/ApplicationServer/profiles/itsoprofile3/bin/
   startServer.sh wasmember09
   ```

   For Stop Command, enter the path the **`stopServer.sh`** command. In our scenario, the path is as follows:

```
/global/scudb/WebSphere/ApplicationServer/profiles/itsoprofile3/bin/
stopNode.sh
```

After selecting the command, add the name of the application server to be stopped at the end of the Stop Command field. In this example, we add wasmember09 to the end of the command:

```
/global/scudb/WebSphere/ApplicationServer/profiles/itsoprofile3/bin/
stopServer.sh wasmember09
```

Leave the Probe Command field blank.

Click **Configure.**

6. Click **OK** on the success box.
7. After creation of the package, click **Cancel** on the SunPlex Agent Builder window.
8. Open a command window and change to your Working Directory, then change to the IBMWASA9/pkg directory. In our example you need to be in

   /workarea/agentbuilder3/IBMWASA9/pkg

   Run the **pkgadd** command to add the newly created IBMWASA9 package:

   ```
   pkgadd -d . IBMWASA9
   ```

9. Zip the agentbuilder3 directory and transfer it to the second system (SHost02). Unzip into the same directory structure, change to the IBMWASA9/pkg directory and run the **pkgadd** command from step 8 to create the package on this system as well:

   ```
   pkgadd -d . IBMWASA9
   ```

Adding an application server resource to Sun Cluster

Using the **scsetup** GUI and **scrgadm** command, add an application server resource to Sun Cluster:

1. On SHost01, run the **scsetup** GUI:

   ```
   /usr/cluster/bin/scsetup
   ```

2. On the Main Menu, select **2) Resource groups**.
3. On the Resource Group Menu, select **3) Add a data service resource to a resource group**.
4. On the Add a Data Service Resource to a Resource Group panel, enter `yes` when prompted to continue. The following steps, 5 to 22 on page 512, are all performed on this panel.
5. Enter the existing resource group when prompted for the group to which you want to add the new resource.

To find the name of existing resource groups, run the **scstat** command in another window. Look for the Resource Groups and Resources section and the Group Name column. Sample output for our scenario is shown in Example 13-14.

Example 13-14 Existing resource groups and resources

```
-- Resource Groups and Resources --

            Group Name          Resources
            ----------          ---------
 Resources: db2_db2inst1_0-rg suncluster db2_db2inst1_0-rs
 Resources: cluster1TM          -
 Resources: cluster2TM          -
```

Here we find our Resource group called db2_db2inst1_0-rg.

6. Select **IBM.WASA9 WASA9 server for Sun Cluster** when prompted for the type of resource that you want to add. This is the resource type that was created in “Creating an application server package” on page 509.
7. Still on the same panel, answer yes when prompted whether the software for this service is installed on each node.
8. Answer yes when prompted to register this resource type. The panel generates and runs the following command to add the resource type:

   ```
   scrgadm -a -t IBM.WASA9
   ```

9. Press Enter to continue.
10. When prompted for the name of the resource that you want to add, enter a name for the resource. Our resource name is wasmem09-rs.
11. When you see the section shown in Example 13-15, enter yes to override the default setting.

Example 13-15 Adding an application server port for monitoring

```
This data service uses the "Port_list" property. The default
    "Port_list" for this data service is as follows:

        <NULL>

    Please check the documentation for this resource type for more
    information on how the list should be set for this resource.

    Do you want to override the default (yes/no) [no]?
```

12. Fill in an application server port. The probe monitoring system from Sun Cluster tries to connect to this port to determine whether the application server is running. We use the application server’s SOAP port.

To find the application server's SOAP port, open the WebSphere Administrative Console. Select **Servers** → **Applications servers** → **wasmember09** → **Ports**. Open **SOAP_CONNECTOR_ADDRESS**.

The SOAP port number for our application server is 8879, thus, for Port number (Ctrl+D to finish), enter 8879.

13. Enter yes to the question whether this is a TCP port.
14. Press Ctrl+D to stop entering ports.
15. The panel now confirms the TCP port just entered, answer **yes** if it is correct.
16. Press Enter to continue.
17. When prompted for extension properties that you would like to set, enter no. However, if you know that you do want to change properties, you can enter yes and change properties. The list of properties might look similar to that shown in Example 13-16.

Example 13-16 Extension properties that can be updated

```
Here are the extension properties for this resource:
        Property Name             Default Setting
         =============             ===============
        Confdir_list              <NULL>
         Monitor_retry_count       4
         Monitor_retry_interval    2
         Probe_timeout             30
         Child_mon_level           -1
       Please enter the list of properties you want to set:
       (Type Ctrl-D to finish  OR  "?" for help)
```

18. Answer yes when prompted to proceed with the update if you are ready to add the application server resource.
19. Based on your entries, the Add a Data Service Resource to a Resource Group panel generates and runs this command:

```
scrgadm -a -j wasmem09-rs -g db2_db2inst1_0-rg -t IBM.WASA9 -y
Scalable=false -y Port_list=8879/tcp
```

If something goes wrong with adding the resource, you can copy and reuse this command instead of stepping through **scsetup** again.

20. Press Enter to continue.
21. Answer no when prompted to enable this resource.
22. Enter q twice to exit **scsetup**.
23. The next step is to update the Thorough_Probe_Interval and Retry_Interval properties. Updating these properties allows the application server enough time to start before Sun Cluster checks to see if it is running. If Sun Cluster

probes too soon, it tries to stop and restart the application server in the middle of its initial start process. The Retry_Interval must be equal to or greater than the Thorough_Probe_Interval multiplied by the Retry_Count. To update these properties:

a. Update the Retry_Interval to 1000:

```
scrgadm -c -j wasmem09-rs -y Retry_Interval=1000
```

b. Update the Thorough_Probe_Interval:

```
scrgadm -c -j wasmem09-rs -y Thorough_Probe_Interval=500
```

These values assume the Retry_Count is set to 2.

24. In a command window, activate the resource group using the **scswitch** command:

```
scswitch -e -j wasmem09-rs
```

Review the application server's log for a successful start. If there are problems starting, you can stop the resource using the **-n** parameter on **scswitch**:

```
scswtich -n -j wasmem09-rs
```

25. Use the **scstat** command to review the status of the resources. The wasmem09-rs resource should be added to the previously selected resource group under the section Resource Groups and Resources as shown in Example 13-17.

Example 13-17 Resources including the wasmem09-rs resource

```
-- Resource Groups and Resources --

            Group Name          Resources
            ----------          ---------
 Resources: db2_db2inst1_0-rg suncluster db2_db2inst1_0-rs nodeagent-rs
wasmem09-rs
 Resources: cluster1TM          -
 Resources: cluster2TM          -
```

Sun Cluster now controls the application server, wasmember09. Sun Cluster is able to stop and start the application server and checks the SOAP port for its status.

Adding the second cluster member

Repeat the steps from 13.4.4, "Configure Sun Cluster to run application server" on page 509 to create a package, resource type, and resource for wasmember10.

13.4.5 Testing Node Agent and application server failover

After configuring the Node Agent and application servers with Sun Cluster, verify that they will successfully failover. The application should continue to run uninterrupted if the Node Agent goes down. If an application server fails, traffic should be rerouted to another member in the cluster until the application server restarts.

Two possible tests include:

- Killing the server processes and ensuring that Sun Cluster restarts them and taking down a machine.
- Verifying that the Deployment Manager remains in action on the second machine.

Failing the Node Agent

To verify that Sun Cluster can monitor and restart the Node Agent, you can kill the Node Agent process:

1. On SHost01, use **scstat** to check where the resource group is located. If it is active on SHost02, use the **scswitch** command to swap the group to SHost01:

   ```
   scswitch -Z -g db2_db2inst1_0-rg
   ```

2. Verify that the Node Agent is started. Either open the Administrative Console and check the Node Agent's status or review the log for the following message:

   ```
   WSVR0001I: Server nodeagent open for e-business.
   ```

   If the Node Agent is not running, start it using the **scswitch** command:

   ```
   scswitch -e -j nodeagent-rs
   ```

3. Check the Node Agent's log. It should synchronize successfully with the Deployment Manager. Look for the following message:

   ```
   ADMS0003I: The configuration synchronization completed successfully.
   ```

4. Find the process number of the Node Agent by looking at the .pid file in the Node Agent's log directory. Use the **kill** command to end the process:

   ```
   kill -9 pid
   ```

5. Use the Trade 6 application. The application should not be affected by the Node Agent failure.
6. Review the Node Agent's log to verify that it is restarting.
7. Verify that it discovers the Deployment Manager process and the application servers, as shown in Example 13-18 on page 515.

Example 13-18 Node Agent discovers Deployment Manager and application servers

```
[4/19/05 15:58:26:894 CDT] 00000032 DiscoveryMBea I   ADMD0023I: The system
discovered process (name: dmgr, type: DeploymentManager, pid: 21791)
[4/19/05 15:58:29:849 CDT] 00000038 DiscoveryMBea I   ADMD0023I: The system
discovered process (name: wasmember09, type: ManagedProcess, pid: 2441)
[4/19/05 15:58:31:161 CDT] 00000041 DiscoveryMBea I   ADMD0023I: The system
discovered process (name: wasmember10, type: ManagedProcess, pid: 2440)
```

8. Trade 6 should continue to work normally.

Failing an application server

To verify that Sun Cluster can monitor and restart an application server, you can kill an application server process as follows:

1. On SHost01, use **scstat** to check where the resource group is located. If it is active on SHost02, use the **scswitch** command to swap the group to SHost01:

   ```
   scswitch -Z -g db2_db2inst1_0-rg
   ```

2. Verify that the Node Agent is started. Either open the Administrative Console and check the Node Agent's status or review the log for the following message:

   ```
   WSVR0001I: Server nodeagent open for e-business.
   ```

   If the Node Agent is not running, start it using the **scswitch** command:

   ```
   scswitch -e -j wasmem09-rs
   ```

3. Run Trade 6 against the wasmember09's WebContainer Inbound Chain and the external Web server. Verify that you can buy stock.
4. Find the process number of the application server by looking at the .pid file in the application server's log directory. Use the kill command to end the process:

   ```
   kill -9 pid
   ```

5. Continue trading with the Trade 6 application via the Web server. The application should not be affected and should route all requests to the other cluster member, wasmember10.
6. Review wasmember09's logs for a successful server start.

7. Verify that it discovers the Node Agent, as shown in Example 13-19.

Example 13-19 Application server discovers Node Agent and finishes start

```
[4/19/05 16:06:15:914 CDT] 00000047 DiscoveryMBea I   ADMD0023I: The system
discovered process (name: nodeagent, type: NodeAgent, pid: 24575)
[4/19/05 16:06:29:852 CDT] 0000000a WsServerImpl  A   WSVR0001I: Server
wasmember09 open for e-business
```

8. Run Trade 6 against wasmember09 directly and the Web server. Trade 6 should work normally. It might recover transactions that were interrupted at failover if you use the Asynchronous_2-Phase Order-Processing Mode for Trade 6. Look for the Transaction recovery messages, such as the following:

```
WTRN0027I: Transaction service recovering 1 transaction.
```

Rebooting the machine

To test that the Node Agent is highly available on both machines using a failover configuration with Sun Cluster, reboot the machine acting as the primary by following these steps:

1. On SHost01, use **scstat** to check where the resource group is located. If it is active on SHost02, use the **scswitch** command to swap to SHost01:

```
scswitch -Z -g db2_db2inst1_0-rg
```

2. Verify that the Node Agents and application servers are started. Open the Administrative Console and check their status. If they are not running, start them using the appropriate **scswitch** commands:

```
scswitch -e -j nodeagent-rs
scswitch -e -j wasmem09-rs
scswitch -e -j wasmem10-rs
```

3. Run Trade 6 to verify that it is working. Use the Web server.
4. Reboot SHost01.
5. Ping the logical host name, suncluster.ibmredbook.com. It should be available shortly.
6. Review the application server logs for a successful restart as shown in Example 13-19.
7. Run Trade 6 again. It might recover transactions interrupted at failover. Look for the Transaction recovery message:

```
WTRN0027I: Transaction service recovering 1 transaction.
```

13.4.6 Troubleshooting

If you encounter problems where Sun Cluster attempts to start, stop, and restart the application server or Node Agent repeatedly, you might need to adjust some of the time related properties on the resource. Sun Cluster might be trying to probe the resource too soon. Try adjusting the Thorough_Probe_Interval and Retry_Interval. There are also timeouts on the stop and start commands.

13.5 Transaction Manager and messaging engine failover with No Operation policy

A discussion of configuring the Transaction Manager failover with No Operation Policy is provided in 9.6, “Transaction Manager failover with No Operation policy” on page 313. A discussion of configuring the messaging engine failover with No Operation Policy is provided in 9.7, “Default messaging provider failover with No Operation policy” on page 347.

In this section, we discuss how to configure Sun Cluster to correctly failover the Transaction Manager and messaging engine with the No Operation Policy. For this scenario, we need an Active/Active (or scalable) configuration. Some of the steps for making the Node Agent and application servers available in an Active/Active scenario are similar to the Active/Passive scenario. A scalable group still requires a failover group which contains a logical and scalable host name. This example uses the db2_db2inst1_0-rg as the failover group containing the logical host name, suncluster.

When using an Active/Active or scalable configuration, WebSphere’s Transaction Manager (TM) and default messaging provider need to be controlled differently. They must be activated manually to avoid conflicts between two active servers trying to use them at the same time.

13.5.1 Additional Sun Cluster setup

To create scalable resources, the Sun Cluster configuration needs a shared address and scalable resource group. The shared address goes in the failover group. The scalable group, which uses the shared address, has a dependency on the failover group.

Adding a shared address to a failover group

The shared host name must be on the same subnet as the system IP addresses and the logical host name. In this example, the logical host name is suncluster and part of the db2_db2inst1_0-rg failover resource group. The shared host

name, a-suncluster, is added to the same group. To add a shared address to a failover group:

1. Add the shared host name to /etc/hosts on the Sun Cluster nodes.
2. Use the **scrgadm** command to create a shared address resource and add it to the existing failover group, db2_db2inst1_0-rg. In this example, the name of the shared host name group is a-suncluster:

```
scrgadm -a -S -j a-suncluster -g db2_db2inst1_0-rg -l a-suncluster
```

3. Activate the new shared address resource using the **scswitch** command:

```
scswitch -e -j a-suncluster
```

For more information about adding a shared address, see the article *How to Add a Shared Address Resource to a Resource Group*, which is available at:

```
http://docs.sun.com/app/docs/doc/817-6564/6mlunk5a0?a=view
```

Creating a scalable resource group

A scalable resource group must be created. It depends on the failover group that contains a shared address. Use the **scrgadm** command to add the resource group. In this case, we have two Sun Cluster nodes, and we want them to both be primaries. The resource group is named websphere_active-rg:

```
scrgadm -a -g websphere_active-rg -y Maximum_Primaries=2 -y
Desired_Primaries=2 -y RG_mode=Scalable -y
RG_dependencies=db2_db2inst1_0-rg
```

Then, switch the group to managed using the **scswitch** command:

```
scswitch -o -g websphere_active-rg
```

Finally, activate the resource group with the **scswitch** command:

```
scswitch -Z -g websphere_active-rg
```

For more information about creating scalable resource groups, see the article *How to Create a Scalable Resource Group*, which is available at:

```
http://docs.sun.com/app/docs/doc/817-6564/6mlunk59s?a=view
```

Creating additional failover group

Part of the application server is on a separate failover resource group. Create an additional group with the **scrgadm** command:

```
scrgadm -a -g websphere-tm-rg -y RG_dependencies=websphere_active-rg
```

In this example, it is called websphere_tm-rg. It has a dependency on the scalable resource group.

13.5.2 Configuring the Deployment Manager

On a remote system, install WebSphere Application Server Network Deployment V6.0.1. Create a Deployment Manager profile and start the Deployment Manager.

13.5.3 Installing the node

To install the node:

1. Install WebSphere Network Deployment 6.0.1 onto the shared disk drive on SHost01.
2. Create a custom profile. Enter the logical host name of the Sun Cluster setup in the Hostname field. For our scenario, we use a-suncluster.ibmredbook.com.
3. Federate the node to the Deployment Manager on the remote machine.

13.5.4 Completing the configuration

To finish the configuration, you need to configure a Web server, create a cluster and cluster members, install the sample application, change settings for the application servers (monitoring policy and core group settings), and enable security if desired.

Configuring a Web server

Install a Web server. In this example, we installed IBM HTTP Server and configured it in the cell using the name http1. Refer to Chapter 6 of IBM *WebSphere V6 Scalability and Performance Handbook*, SG24-6392 for information about how to configure a Web server in a cell.

Installing Trade 6

Install Trade 6 on the Deployment Manager. Follow the directions that are included in the Trade 6 download package for setup and install, using the trade.jacl script for the setup. For additional information about how to install and configure Trade 6, refer to Chapter 8 of *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392.

During the installation, create an application server cluster with two cluster members. In our example, we used the following names:

- Cluster name: `wascluster01`
- Cluster members: `wasmember09, wasmember10`

Map the Trade 6 Web modules to the previously created Web server, http1.

Changing application server monitoring policies

Change the automatic restart property of the application servers. This allows Sun Cluster to gain control over restarting the servers and prevents the Node Agent and Sun Cluster from trying to restart an application server at the same time.

To disable the automatic restart for the application servers:

1. In the Administrative Console, go to **Servers** → **Application servers** → **wasmember09** → **Java and Process Management** → **Process Definition** → **Monitoring Policy**.
2. Deselect the Automatic restart field. Click **OK** and save your changes.
3. Repeat this for wasmember10.
4. Restart the application servers and the Node Agent.

You can find more information about this option in the InfoCenter article *Monitoring policy settings*.

Security considerations

Enable security if necessary. Use the soap.client.props properties file on the node to enter the user name and password for stopping the Node Agent and application servers. If you do not add the user name and password to the soap.client.properties file, the **stopNode** and **stopServer** commands expects the properties on the command line, which does not appear as part of the Sun Cluster start and stop scripts.

Encode the soap.client.props property file by following these steps:

1. Go to the node's profile directory. Under the properties directory, open soap.client.props.
2. Add the user name and password to the JMX SOAP connector identity section:
   - com.ibm.SOAP.loginUserid=*username*
   - com.ibm.SOAP.loginPassword=*password*
3. Use the PropFilePasswordEncoder tool to encode the login password. Pass com.ibm.SOAP.loginPassword as the property to encode. From the node's profile bin directory, run the following:

   ```
   ./PropFilePasswordEncoder.sh ../properties/soap.client.props
   com.ibm.SOAP.loginPassword
   ```

   After encoding, the properties file might be rearranged. A backup properties file will also be created under the name of soap.client.props.bak.

Also see the InfoCenter article *Protecting plain text passwords*.

Changing core group policies

You need to add or change the transaction policy for servers running as scalable. If the servers might run messaging engines, you also have to add or change the policy for the messaging engines. In both of these cases, the policy types are No Operation or NoOP policies, which means that WebSphere does not start the Transaction Manager or messaging engine.

For our example, a NoOP policy for wasmember09's Transaction Manager is created with a match criteria of the following:

- GN_PS=wascell01\wasna04\wasmember09
- IBM_hc=wascluster01
- type=WAS_TRANSACTIONS

A NoOP policy is also created for the messaging engine, wascluster01.005-wascluster01 with a match criteria of the following:

- WSAF_SIB_MESSAGING_ENGINE=wascluster01.005-wascluster01
- type=WSAF_SIB

See 9.6.2, "Transaction Manager with No Operation policy scenario" on page 316 and 9.7.2, "Default messaging provider with No Operation policy scenario" on page 348, where we describe the concept and how to set up the policies in detail.

Testing the application

Start the Trade 6 application and verify that it works correctly by populating the Trade 6 database and trading. Stop the Node Agent and cluster.

13.5.5 Configuring the Node Agent with Sun Cluster

Configuring the Node Agent to run as scalable is similar to the failover mode. There are some changes to the setup, but a resource type and resource must still be created. When the resource is enabled, the Node Agent essentially is running on both systems. You see duplicate messages in the logs and there is a Java process running on both systems.

Creating a Node Agent package

Use the SunPlex Agent Builder to create a package with start and stop commands for the Node Agent. To create a Node Agent package:

1. Open SunPlex Agent Builder:

   ```
   /usr/cluster/bin/scdsbuilder
   ```

2. In the Step 1 of the SunPlex Agent Builder window, fill in these fields:
   - Vendor Name: `IBM`

- Application Name: `AWASNA`
- Working Directory: `/workarea/agentbuilder5`
- Scalable: selected
- Resource Type: *ksh*

Click **Create**.

3. Click **OK** in the success window.
4. Click **Next** on the SunPlex Agent Builder window.
5. In Step 2 of the SunPlex Agent Builder window, fill in the start and stop commands for the Node Agent in the appropriate fields.

   For Start Command, enter the path to the **startNode.sh** command. In our example the path is:

   ```
   /global/scudb/WebSphere/ApplicationServer/profiles/itsoprofile4/bin/
   startNode.sh
   ```

   For Stop Command, enter the path to the **stopNode.sh** command. In our example the path is:

   ```
   /global/scudb/WebSphere/ApplicationServer/profiles/itsoprofile4/bin/
   stopNode.sh
   ```

   Leave the Probe Command blank.

   Click **Configure**.
6. Click **OK** on the success box.
7. After creation of the package, click **Cancel** on the SunPlex Agent Builder window.
8. Open a command window and change to your Working Directory, then change to the IBMWASNA/pkg directory. In our example you need to be in

   /workarea/agentbuilder5/IBMAWASNA/pkg

   Run the **pkgadd** command to add the newly created IBMAWASNA package:

   ```
   pkgadd -d . IBMAWASNA
   ```
9. Zip the agentbuilder5 directory and transfer it to the second system (SHost02). Unzip into the same directory structure, change to the IBMWASNA/pkg directory and run the **pkgadd** command from step 8 to create the package on this system as well:

   ```
   pkgadd -d . IBMAWASNA
   ```

Adding a Node Agent resource to Sun Cluster

Using the **scsetup** GUI and **scrgadm** command, add a Node Agent resource to Sun Cluster:

1. On SHost01, run the **scsetup** GUI:

```
/usr/cluster/bin/scsetup
```

2. On the Main Menu, select **2) Resource groups**.
3. On the Resource Group Menu, select **3) Add a data service resource to a resource group**.
4. On the Add a Data Service Resource to a Resource Group panel, enter `yes` when asked to continue. The following steps, 5 to 25 on page 525, are all performed on this panel.
5. Enter the existing resource group when prompted for the group that you want to add the new resource. Enter the name of the scalable resource group. In this example, we use websphere_active-rg.
6. Select **IBM.AWASNA AWASNA server for Sun Cluster** when prompted to select the type of resource that you want to add. This is the resource type that was created in "Creating a Node Agent package" on page 521.
7. Still on the same panel, answer `yes` to the question whether the software for this service is installed on each node.
8. Answer `yes` to the question if it is okay to register this resource type now. The panel generates and runs the following command to add the resource type:

```
scrgadm -a -t IBM.AWASNA
```

9. Press Enter to continue.
10. When prompted for the name of the resource that you want to add, enter a name for the resource. Our resource name is `nodeagent-rs`.
11. Next, you are requested to specify SharedAddress resources as shown in Example 13-20. This is the shared address added to the failover group in "Adding a shared address to a failover group" on page 517. For our example this is a-suncluster.

Example 13-20 Adding SharedAddress resources

```
For scalable resources, you must specify a list of SharedAddress
    resources upon which the new scalable resource depends. Since network
    resources can only belong to failover resource groups, the
    SharedAddress resources that you name must live in a different
    resource group.
   Please list one SharedAddress resource dependency per line. At least
    one such resource must be given. When finished, type Control-D.

   SharedAddress resource:
```

12. Press Ctrl+D to stop entering shared addresses.
13. The panel confirms the list of shared address. Enter `yes` if it is correct.

14. When you see the section shown in Example 13-21, type `yes` to override the default setting.

Example 13-21 Adding a Deployment Manager port for monitoring

```
This data service uses the "Port_list" property. The default
    "Port_list" for this data service is as follows:

        <NULL>

    Please check the documentation for this resource type for more
    information on how the list should be set for this resource.

    Do you want to override the default (yes/no) [no]?
```

15. Fill in a Node Agent port. The probe monitoring system from Sun Cluster tries to connect to this port to determine whether the Node Agent is running. We use the Node Agent's SOAP port.

    To find the Node Agent's SOAP port, open the WebSphere Administrative Console. Select **System administration → Node agents → nodeagent → Ports**. Open **SOAP_CONNECTOR_ADDRESS**.

    The SOAP port number for our Node Agent is 8880, thus, for Port number (Ctrl+D to finish), enter 8880.

16. Enter `yes` to the question whether this is a TCP port.

17. Press Ctrl+D to stop entering ports.

18. The panel now confirms the TCP port just entered. Answer `yes` if it is correct.

19. Press Enter to continue.

20. When prompted for any extension properties that you would like to set, enter `no`. However, if you know that you do want to change properties, you can enter `yes` and change properties. The list of properties might look similar to that shown in Example 13-22.

Example 13-22 Extension properties that can be updated

```
Here are the extension properties for this resource:
      Property Name             Default Setting
       =============             ===============
      Confdir_list              <NULL>
       Monitor_retry_count       4
       Monitor_retry_interval    2
       Probe_timeout             30
       Child_mon_level           -1
   Please enter the list of properties you want to set:
      (Type Ctrl-D to finish  OR  "?" for help)
```

21. Answer `yes` when prompted to proceed with the update if you are ready to add the Node Agent resource.

22. Based on your entries, the Add a Data Service Resource to a Resource Group panel generates and runs this command:

```
scrgadm -a -j nodeagent-rs -g websphere_active-rg -t IBM.AWASNA -y
Scalable=true -y Network_resources_used=a-suncluster -y
Port_list=8880/tcp
```

    If something goes wrong with adding the resource, you can copy and reuse this command instead of stepping through **`scsetup`** again.

23. Press Enter to continue.

24. Answer `no` when asked whether you want to enable this resource.

25. Enter `q` twice to exit **`scsetup`**.

26. Update the Thorough_Probe_Interval and Retry_Interval properties.

    Updating these properties allows the Node Agent enough time to start before Sun Cluster checks to see if it is running. If Sun Cluster probes too soon, it tries to stop and restart the Node Agent in the middle of its initial start process. The Retry_Interval must be equal to or greater than the Thorough_Probe_Interval multiplied by the Retry_Count. To update these properties:

    a. Update the Retry_Interval to 1000:

```
scrgadm -c -j nodeagent-rs -y Retry_Interval=1000
```

    b. Update the Thorough_Probe_Interval:

```
scrgadm -c -j nodeagent-rs -y Thorough_Probe_Interval=500
```

    These values assume the Retry_Count is set to 2.

27. Activate the resource using the **`scswitch`** command.

```
scswitch -e -j nodeagent-rs
```

    Review the Node Agent's log for a successful start. If there are problems starting, you can stop the resource using the **`-n`** parameter on **`scswitch`**:

```
scswtich -n -j nodeagent-rs
```

28. Use the **scstat** command to review the status of the resources. The nodeagent-rs resource should be added to the previously selected resource group under the Resource Groups and Resources heading as listed in Example 13-13 on page 508.

Example 13-23 Resources including the nodeagent-rs resource

```
-- Resource Groups and Resources --

            Group Name          Resources
            ----------          ---------
 Resources: db2_db2inst1_0-rg   suncluster db2_db2inst1_0-rs a-suncluster
 Resources: websphere_active-rg nodeagent-rs
 Resources: cluster1TM          -
 Resources: cluster2TM          -
```

Sun Cluster can now control the Node Agent. It is able to stop and start the Node Agent and checks the SOAP port for its status.

13.5.6 Configuring an application server with Sun cluster

Configuring the application server to run as scalable requires several additional steps than setting up for failover. The application server, the Transaction Manager and the messaging engine, are each controlled with separate resources. When the resource is enabled, essentially, the application server is started on both systems. You see duplicate messages in the logs, and there is a Java process running on both systems. The application server is started only fully on one system. The mirrored server is started partially and waits for Transaction Manager start.

Preparing start and stop scripts for the application server

The scripts needed are explained in detail in “Scripts to start, stop, and monitor WebSphere resources” on page 331 for the Transaction Manager and “Scripts to start, stop, and monitor WebSphere resources” on page 357 for the messaging engine failover.

You need to make the following changes the scripts for them to work on Sun Cluster:

1. In the wasctrl-as script, update the return codes near the top of the script as follows. For Sun Cluster, a return code of zero is successful.
   - UNKNOWN=2
   - ONLINE=0
   - OFFLINE=1

2. Depending on the speed of your system, adjust the **-timeout** value on the **startServer.sh** and **stopServer.sh** commands in the start and stop sections of the script.
3. In the status section, change the **netstat** command to the following:

```
netstat -anP tcp | grep ${AS_TCPP} | grep LISTEN
```

4. Gather the following information for use with the script:
   - The node profile directory: /global/WebSphere/ApplicationServer/profiles/itsoprofile
   - The SOAP port for wasmember09: 8879
   - The name of the application server: wasmember09

Creating an application server package

Use the SunPlex Agent Builder to create a package with start and stop commands for an application server. The start and stop commands come from the wasctrl-as script. To create an application server package:

1. Open SunPlex Agent Builder:

```
/usr/cluster/bin/scdsbuilder
```

2. In the Step 1 of the SunPlex Agent Builder window, fill in these fields:
   - Vendor Name: `IBM`
   - Application Name: `WASA9`
   - Working Directory: `/workarea/agentbuilder3`
   - Scalable: selected
   - Resource Type: *ksh*

   Click **Create**.
3. Click **OK** on the success window.
4. Click **Next** on the SunPlex Agent Builder window.
5. In the Step 2 of the SunPlex Agent Builder window, fill in the start and stop commands for the application server in the appropriate fields.

   For Start Command, enter the path to the wasctrl-as script. In our example the path is /global/scudb/scripts/wasctrl-as. After selecting the command, add the rest of the script parameters to the Start Command field. In this example, we add the start command, the profile directory, the SOAP port, and the server name to the end of the command:

```
/global/scudb/scripts/wasctrl-as start
/global/WebSphere/ApplicationServer/profiles/itsoprofile 8879
wasmember09
```

   For Stop Command, enter the path the wasctrl-as script again. In this example, we use /global/scudb/scripts/wasctrl-as. After selecting the

command, add the rest of the script parameters to the Stop Command field. In this example, we add the stop command, the profile directory, the SOAP port and the server name to the end of the command:

```
/global/scudb/scripts/wasctrl-as stop
/global/WebSphere/ApplicationServer/profiles/itsoprofile 8879
wasmember09
```

For Probe Command, enter the path the wasctrl-as script again. In this example, we use /global/scudb/scripts/wasctrl-as. After selecting the command, add the rest of the script parameters to the Probe Command field. In this example, we add the status command, the profile directory, the SOAP port and the server name to the end of the command:

```
/global/scudb/scripts/wasctrl-as status
/global/WebSphere/ApplicationServer/profiles/itsoprofile 8879
wasmember09
```

Click **Configure.**

6. Click **OK** on the success box.
7. After creation of the package, click **Cancel** on the SunPlex Agent Builder window.
8. Open a command window and change to your Working Directory, then change to the IBMWASA9/pkg directory. In our example, you need to be in the following directory:

   /workarea/agentbuilder3/IBMWASA9/pkg

   Run the **pkgadd** command to add the newly created IBMWASA9 package.

   ```
   pkgadd -d . IBMWASA9
   ```

9. Zip the agentbuilder3 directory and transfer it to the second system (SHost02). Unzip into the same directory structure, change to the IBMWASA9/pkg directory and run the **pkgadd** command from step 8 to create the package on this system as well:

   ```
   pkgadd -d . IBMWASA9
   ```

Adding an application server resource to Sun Cluster

Using the **scsetup** GUI and **scrgadm** command, add an application server resource to Sun Cluster:

1. On SHost01, run the **scsetup** GUI:

   ```
   /usr/cluster/bin/scsetup
   ```

2. On the Main Menu, select **2) Resource groups**.
3. On the Resource Group Menu, select **3) Add a data service resource to a resource group**.

4. On the Add a Data Service Resource to a Resource Group panel, enter `yes` when prompted to continue. The following steps, 5 to 23 on page 531, are all performed on this panel.
5. Enter the existing resource group when prompted for the group for which you want to add the new resource. Enter the name of the scalable resource group. In this example, we use `websphere_active-rg`.
6. Select **IBM.WASA9 WASA9 server for Sun Cluster** when prompted for the type of resource that you want to add. This is the resource type that was created in “Creating an application server package” on page 527.
7. Still on the same panel, answer `yes` to the question whether the software for this service is installed on each node.
8. Answer `yes` to the question whether it is okay to register this resource type now. The panel generates and runs the following command to add the resource type:

   ```
   scrgadm -a -t IBM.WASA9
   ```

9. Press Enter to continue.
10. When prompted for the name of the resource that you want to add, enter a name for the resource. Our resource name is `wasmem09-rs`.
11. Specify SharedAddress resources as shown in Example 13-24. This is the shared address added to the failover group in “Adding a shared address to a failover group” on page 517. For our example this is a-suncluster.

Example 13-24 Adding SharedAddress resources

```
For scalable resources, you must specify a list of SharedAddress
    resources upon which the new scalable resource depends. Since network
    resources can only belong to failover resource groups, the
    SharedAddress resources that you name must live in a different
    resource group.

    Please list one SharedAddress resource dependency per line. At least
    one such resource must be given. When finished, type Control-D.

    SharedAddress resource:
```

12. When you see the section that is shown in Example 13-25, type `yes` to override the default setting.

Example 13-25 Adding an application server port for monitoring

```
This data service uses the "Port_list" property. The default
    "Port_list" for this data service is as follows:

        <NULL>

    Please check the documentation for this resource type for more
    information on how the list should be set for this resource.

    Do you want to override the default (yes/no) [no]?
```

13. Fill in an application server port. The probe monitoring system from Sun Cluster tries to connect to this port to determine whether the application server is running. We use the application server's SOAP port.

    To find the application server's SOAP port, open the WebSphere Administrative Console. Select **Servers** → **Applications servers** → **wasmember09** → **Ports**. Open **SOAP_CONNECTOR_ADDRESS**.

    The SOAP port number for our application server is 8879, thus, for Port number (Ctrl+D to finish), enter 8879.

14. Enter `yes` to the question whether this is a TCP port.

15. Press Ctrl+D to stop entering ports.

16. The panel now confirms the TCP port just entered, answer `yes` if it is correct.

17. Press Enter to continue.

18. When prompted for extension properties that you would like to set, enter `no`. However, if you know that you do want to change properties, you can enter `yes` and change properties. The list of properties might look similar to that shown in Example 13-26.

Example 13-26 Extension properties that can be updated

```
Here are the extension properties for this resource:
       Property Name             Default Setting
        =============             ===============
       Confdir_list              <NULL>
        Monitor_retry_count       4
        Monitor_retry_interval    2
        Probe_timeout             30
        Child_mon_level           -1
      Please enter the list of properties you want to set:
      (Type Ctrl-D to finish  OR  "?" for help)
```

19. Answer `yes` when prompted to proceed with the update if you are ready to add the application server resource.

20. Based on your entries, the Add a Data Service Resource to a Resource Group panel generates and runs this command:

```
scrgadm -a -j wasmem09-rs -g db2_db2inst1_0-rg -t IBM.WASA9 -y
Scalable=true -y Port_list=8879/tcp
```

    If something goes wrong with adding the resource, you can copy and reuse this command instead of stepping through **`scsetup`** again.

21. Press Enter to continue.

22. Answer `no` when prompted to enable this resource.

23. Enter `q` twice to exit **`scsetup`**.

24. The next step is to update the Thorough_Probe_Interval and Retry_Interval properties.

    Updating these properties allows the application server enough time to start before Sun Cluster checks to see if it is running. If Sun Cluster probes too soon, it tries to stop and restart the application server in the middle of its initial start process. The Retry_Interval must be equal to or greater than the Thorough_Probe_Interval multiplied by the Retry_Count. To update these properties:

    a. Update the Retry_Interval to 1000:

```
scrgadm -c -j wasmem09-rs -y Retry_Interval=1000
```

    b. Update the Thorough_Probe_Interval:

```
scrgadm -c -j wasmem09-rs -y Thorough_Probe_Interval=500
```

    These values assume the Retry_Count is set to 2.

25. Activate the resource group using the **`scswitch`** command:

```
scswitch -e -j wasmem09-rs
```

    Review the application server's log for a successful start. If there are problems starting, you can stop the resource using the **`-n`** parameter on **`scswitch`**:

```
scswtich -n -j wasmem09-rs
```

26. Use the **`scstat`** command to review the status of the resources. The wasmem09-rs resource should be added to the previously selected resource

group under the section Resource Groups and Resources as shown in Example 13-27.

Example 13-27 Resources including the wasmem09-rs resource

```
-- Resource Groups and Resources --

            Group Name          Resources
            ----------          ---------
 Resources: db2_db2inst1_0-rg   suncluster db2_db2inst1_0-rs a-suncluster
 Resources: websphere_active-rg nodeagent-rs wasmem09-rs
 Resources: websphere_tm-rg     -
 Resources: cluster1TM          -
 Resources: cluster2TM          -
```

Sun Cluster now controls the application server, wasmember09. Sun Cluster can stop and start the application server and check the SOAP port for its status. The server is started to the point of when the Transaction Manager would normally start. Because there is a NoOP policy for the Transaction Manager, the server does not finish starting until the Transaction Manager is started manually.

Adding the second cluster member

Repeat the steps to create a package, resource type, and resource for wasmember10.

Preparing start and stop scripts for the Transaction Manager

To finish the application server start, the Transaction Manager needs to be started manually. This is done through several scripts and the HAMonitor application. The wasctrl-tm script calls stop, start, and monitor scripts for the Transaction Manager. For more information about the scripts and the HAMonitor application, refer to 9.6, “Transaction Manager failover with No Operation policy” on page 313.

1. In the wasctrl-tm script, update the return codes near the top of the script. For Sun Cluster, a return code of zero is successful.
   - UNKNOWN=2
   - ONLINE=0
   - OFFLINE=1
2. In the start section, change the **netstat** command to

```
netstat -anP tcp | grep ${AS_TCPP} | grep LISTEN
```

3. At the end of each Perl command, add the HOST_NAME parameter, ${HOST_NAME} as shown in Example 13-28.

Example 13-28 Adding HOST_NAME to the Perl command in the start section

```
RetC=`perl $SCRIPT_PATH/activategroup.pl ${CELL_NAME} ${NODENAME}
${SERVER_NAME} ${AS_TCPP} ${CRITERIA} ${HOST_NAME} |awk '{print $1}'`
```

4. On each of the Perl scripts, activategroup.pl, deactivategroup.pl, and monitorgroup.pl, add a HOST_NAME parameter. Replace the existing HOST_NAME parameter with one that is passed into the script, as shown in Example 13-29.

Example 13-29 Example of updating the monitorgroup.pl script

```
if (@ARGV != 6) {
   print "Illegal number of arguments";
} else {
  $CELLNAME= $ARGV[0];
  $NODENAME = $ARGV[1];
  $SERVERNAME = $ARGV[2];
  $PORTNUMBER = $ARGV[3];
  $MATCHING = $ARGV[4];
  $HOST = $ARGV[5]
}

#my $hostname = hostname();
my $getrequest =
'http://'.$HOST.":".$PORTNUMBER."/HAMonitorWeb/QueryLocalMemberState?ms=GN_PS="
.$CELLNAME."\\".$NODENAME."\\".$SERVERNAME.",".$MATCHING."&s=a";
```

5. Gather the following information for use with the wasctrl-tm script:
   - The node profile directory: /global/WebSphere/ApplicationServer/profiles/itsoprofile
   - The default host port of wasmember09 (WC_defaultHost): 9083
   - The host name of wasmember09: a-suncluster
   - The SOAP port of wasmember09 (SOAP_CONNECTOR_ADDRESS): 8879
   - The match criteria for the wasmember09's Transaction Manager NoOP policy, without the GN_PS criteria: IBM_hc=wascluster01,type=WAS_TRANSACTIONS
   - The name of the cell wasmember09 is in: wascell02
   - The server name: wasmember09

- The path to the rest of the Perl scripts and controlHA.pty: /global/scudb/scripts
- The node name that wasmember09 belongs to: wasnode04

Creating a Transaction Manager package

Use the SunPlex Agent Builder to create a package with start and stop commands for the Transaction Manager. The start and stop commands come from the wasctrl-tm script. The Transaction Manager resource is a failover resource. To create an application server package:

1. Open SunPlex Agent Builder:

   ```
   /usr/cluster/bin/scdsbuilder
   ```

2. In the Step 1 of the SunPlex Agent Builder window, fill in these fields:
   - Vendor Name: `IBM`
   - Application Name: `WASTMb`
   - Working Directory: `/workarea/agentbuilder4`
   - Failover: selected
   - Resource Type: *ksh*

   Click **Create**.

3. Click **OK** on the success window.
4. Click **Next** on the SunPlex Agent Builder window.
5. In the Step 2 of the SunPlex Agent Builder window, fill in the start and stop commands in the appropriate fields.

   For Start Command, enter the path to the wasctrl-tm script. In this example, we use global/scudb/scripts/wasctrl-tm. After selecting the command, add the rest of the script parameters to the Start Command field. In this example, we add the start command, the profile directory, the SOAP port, and the server name to the end of the command:

   ```
   /global/scudb/scripts/wasctrl-tm start
   /global/scudb/WebSphere/AppServer/profiles/itsoprofile04/ 9083
   a-suncluster 8879 IBM_hc=wascluster01,type=WAS_TRANSACTIONS wascell02
   wasmember09 /global/scudb/scripts wasna04
   ```

   For Stop Command, enter the path the wasctrl-tm script again. In this example, we use /global/scudb/scripts/wasctrl-tm. After selecting the command, add the rest of the script parameters to the Stop Command field. In this example, we add the stop command, the profile directory, the SOAP port, and the server name to the end of the command:

   ```
   /global/scudb/scripts/wasctrl-tm stop
   /global/scudb/WebSphere/AppServer/profiles/itsoprofile04/ 9083
   a-suncluster 8879 IBM_hc=wascluster01,type=WAS_TRANSACTIONS wascell02
   wasmember09 /global/scudb/scripts wasna04
   ```

Leave the Probe Command field blank.

Click **Configure.**

6. Click **OK** on the success box.
7. After creating the package, click **Cancel** on the SunPlex Agent Builder window.
8. Open a command window and change to your Working Directory, then change to the IBMWASTM/pkg directory. In our example you need to be in

   /workarea/agentbuilder4/IBMWASTM/pkg

   Run the **pkgadd** command to add the newly created IBMWASTM package:

   ```
   pkgadd -d . IBMWASTM
   ```

9. Zip the agentbuilder4 directory and transfer it to the second system (SHost02). Unzip into the same directory structure, change to the IBMWASTM/pkg directory and run the **pkgadd** command from step 8 to create the package on this system as well:

   ```
   pkgadd -d . IBMWASTM
   ```

Adding a Transaction Manager resource

Using the **scsetup** GUI and **scrgadm** command, add a Transaction Manager resource to Sun Cluster:

1. On SHost01, run the **scsetup** GUI:

   ```
   /usr/cluster/bin/scsetup
   ```

2. On the Main Menu, select **2) Resource groups**.
3. On the Resource Group Menu, select **3) Add a data service resource to a resource group**.
4. On the Add a Data Service Resource to a Resource Group panel, enter `yes` when prompted to continue. The following steps, 5 to 22 on page 537, are all performed on this panel.
5. On the Add a Data Service Resource to a Resource Group panel, enter the additional failover resource group, websphere_tm-rg.
6. Select **IBM.WASTM WASTM server for Sun Cluster** when prompted to select the type of resource that you want to add. This is the resource type that was created in “Creating a Transaction Manager package” on page 534.
7. Still on the same panel, answer `yes` to the question whether the software for this service is installed on each node.
8. Answer `yes` to the question if it is okay to register this resource type now. The panel generates and runs the following command to add the resource type:

   ```
   scrgadm -a -t IBM.WASTM
   ```

9. Press Enter to continue.
10. When prompted for the name of the resource that you want to add, enter a name for the resource. Our resource name is `wasmem09TM-rs`.
11. When you see the section shown in Example 13-30, type `yes` to override the default setting.

Example 13-30 Adding an application server port for monitoring

```
This data service uses the "Port_list" property. The default
    "Port_list" for this data service is as follows:

        <NULL>

    Please check the documentation for this resource type for more
    information on how the list should be set for this resource.

    Do you want to override the default (yes/no) [no]?
```

12. Fill in an application server port.

    The probe monitoring system from Sun Cluster tries to connect to this port to determine whether the application server is running. We use the application server's default host port.

    To find the application server's default host port, open the WebSphere Administrative Console. Select **Servers** → **Applications servers** → **wasmember09** → **Ports**. Open **WC_defaulthost**.

    The port number for our application server is 9083. Thus, for Port number (Ctrl+D to finish), enter `9083`.
13. Enter `yes` to the question whether this is a TCP port.
14. Press Ctrl+D to stop entering ports.
15. The panel confirms the TCP port just entered. Answer `yes` if it is correct.
16. Press Enter to continue.

17. When prompted for extension properties that you would like to set , enter `no`. However, if you know that you do want to change properties, you can enter `yes` and change properties. The list of properties might look similar to that shown in Example 13-31.

Example 13-31 Extension properties that can be updated

```
Here are the extension properties for this resource:
       Property Name             Default Setting
        =============             ===============
       Confdir_list              <NULL>
        Monitor_retry_count       4
        Monitor_retry_interval    2
        Probe_timeout             30
        Child_mon_level           -1
      Please enter the list of properties you want to set:
      (Type Ctrl-D to finish  OR  "?" for help)
```

18. Answer `yes` when prompted to proceed with the update if you are ready to add the application server resource.

19. Based on your entries, the Add a Data Service Resource to a Resource Group panel generates and runs this command:

```
scrgadm -a -j wasmem09TM-rs -g websphere_tm-rg -t IBM.WASTM -y
Scalable=false -y Port_list=9083/tcp
```

    If something goes wrong with adding the resource, you can copy and reuse this command instead of stepping through **`scsetup`** again.

20. Press Enter to continue.

21. Answer `no` when prompted to enable this resource.

22. Enter `q` twice to exit **`scsetup`**.

23. Update the Thorough_Probe_Interval and Retry_Interval properties.

    We update these to allow the application server enough time to start before Sun Cluster checks to see if it is running. If Sun Cluster probes too soon, it tries to stop and restart the application server in the middle of its initial start process. The Retry_Interval must be equal to or greater than the Thorough_Probe_Interval multiplied by the Retry_Count. To update these properties:

    a. Update the Retry_Interval to 360:

```
scrgadm -c -j wasmem09-rs -y Retry_Interval=360
```

    b. Update the Thorough_Probe_Interval:

```
scrgadm -c -j wasmem09-rs -y Thorough_Probe_Interval=180
```

    These values assume the Retry_Count is set to 2.

24. Activate the resource group using the **scswitch** command:

```
scswitch -e -j wasmem09TM-rs
```

Review the application server's log for a successful start. If there are problems starting, you can stop the resource using the **-n** parameter on **scswitch**:

```
scswtich -n -j wasmem09TM-rs
```

25. Use **scstat** to review the status of the resources. The wasmem09-rs resource should be added to the previously selected resource group under the section Resource Groups and Resources, as shown in Example 13-32.

Example 13-32 Resources including the wasmem09TM-rs resource

```
-- Resource Groups and Resources --

            Group Name          Resources
            ----------          ---------
 Resources: db2_db2inst1_0-rg   suncluster db2_db2inst1_0-rs s-suncluster
 Resources: websphere_active-rg nodeagent-rs wasmem09-rs
 Resources: websphere_tm-rg     wasmem09TM-rs
 Resources: cluster1TM          -
 Resources: cluster2TM          -
```

Sun Cluster now controls the Transaction Manager for wasmember09 and is able to stop and start the Transaction Manager.

Adding the second cluster member

Repeat these steps to create a package, resource type, and resource for wasmember10.

Preparing start and stop scripts for a messaging engine

The messaging engine also needs to be started manually. This is done through several scripts and the HAMonitor application. The wasctrl-me script calls stop, start, and monitor scripts for the messaging engine. The messaging engine being controlled is wascluster01.005-wascluster01. To prepare these scripts:

1. On the wasctrl-me script, update the return codes near the top of the script. For Sun Cluster, a return code of zero is successful.
   - UNKNOWN=2
   - ONLINE=0
   - OFFLIN=1
2. In the start section, change the **netstat** command to the following:

```
netstat -anP tcp | grep ${AS_TCPP} | grep LISTEN
```

3. At the end of each Perl command, add the HOST_NAME parameter, ${HOST_NAME} as shown in Example 13-33.

Example 13-33 Adding HOST_NAME to the Perl command in the start section

```
RetC=`perl $SCRIPT_PATH/activategroupME.pl  ${AS_TCPP} ${CRITERIA}
${HOST_NAME}|awk '{print $1}'`
```

4. On each of the Perl scripts, activategroupME.pl, deactivategroupME.pl, and monitorgroupME.pl, add a HOST_NAME parameter. Replace the existing HOST_NAME parameter with one that is passed into the script, as shown in Example 13-34.

Example 13-34 Modification to the monitorgroupME.pl script to add hostname parameter

```
if (@ARGV != 3) {
   print "Illegal number of arguments";
} else {
  $PORTNUMBER = $ARGV[0];
  $MATCHING = $ARGV[1];
  $HOST = $ARGV[2]
}

#my $hostname = hostname();
my $getrequest =
'http://'.$HOST.":".$PORTNUMBER."/HAMonitorWeb/QueryLocalMemberState?ms=".$MATC
HING."&s=a";
```

5. Gather the following information for use with the wasctrl-me script:
   - The node profile directory: /global/WebSphere/ApplicationServer/profiles/itsoprofile
   - The default host port of wasmember09 (WC_defaultHost): 9083
   - The host name of wasmember09: a-suncluster
   - The SOAP port of wasmember09 (SOAP_CONNECTOR_ADDRESS): 8879
   - The match criteria for the wasmember09's messaging engine NoOP policy: WSAF_SIB_MESSAGING_ENGINE=wascluster01.005-wascluster01, type=WSAF_SIB
   - The path to the rest of the Perl scripts and controlHA.pty: /global/scudb/scripts

Creating messaging engine package

Use the SunPlex Agent Builder to create a package with start and stop commands for the messaging engine. The start and stop commands come from the wasctrl-me script. The messaging engine resource is a failover resource.

To create an application server package:

1. Open SunPlex Agent Builder:

   ```
   /usr/cluster/bin/scdsbuilder
   ```

2. In the Step 1 of the SunPlex Agent Builder window, fill in these fields:

   – Vendor Name: `IBM`
   – Application Name: `WASME`
   – Working Directory: `/workarea/agentbuilder5`
   – Failover: selected
   – Resource Type: *`ksh`*

   Click **Create**.

3. Click **OK** on the success window.

4. Click **Next** on the SunPlex Agent Builder window.

5. In the Step 2 of the SunPlex Agent Builder window, fill in the start and stop commands in the appropriate fields.

   For Start Command, enter the path to the wasctrl-me script. In this example, we use /global/scudb/scripts/wasctrl-me. After selecting the command, add the rest of the script parameters to the Start Command field. In this example, we add the start command, the profile directory, the SOAP port, and the server name to the end of the command:

   ```
   /global/scudb/scripts/wasctrl-me start
   /global/scudb/WebSphere/AppServer/profiles/itsoprofile04/ 9083
   a-suncluster 8879
   WSAF_SIB_MESSAGING_ENGINE=wascluster01.005-wascluster01,
   type=WSAF_SIB /global/scudb/scripts
   ```

   For Stop Command, enter the path the wasctrl-as script again. In this example, we use /global/scudb/scripts/wasctrl-me. After selecting the command, add the rest of the script parameters to the Stop Command field. In this example, we add the stop command, the profile directory, the SOAP port, and the server name to the end of the command:

   ```
   /global/scudb/scripts/wasctrl-me stop
   /global/scudb/WebSphere/AppServer/profiles/itsoprofile04/ 9083
   a-suncluster 8879
   WSAF_SIB_MESSAGING_ENGINE=wascluster01.005-wascluster01,
   type=WSAF_SIB /global/scudb/scripts
   ```

   Leave the Probe Command blank, and click **Configure**.

6. Click **OK** on the success box.
7. After creation of the package, click **Cancel** on the SunPlex Agent Builder window.
8. Open a command window and change to your Working Directory, then change to the IBMWASME/pkg directory. In our example you need to be in the following directory:

   /workarea/agentbuilder5/IBMWASME/pkg

   Run the **pkgadd** command to add the newly created IBMWASME package:

   ```
   pkgadd -d . IBMWASME
   ```

9. Zip the agentbuilder5 directory and transfer it to the second system (SHost02). Unzip into the same directory structure, change to the IBMWASME/pkg directory and run the pkgadd command from step 8 to create the package on this system as well:

   ```
   pkgadd -d . IBMWASME
   ```

Adding a messaging engine resource

Using the **scsetup** GUI and **scrgadm** command, add a messaging engine resource to Sun Cluster:

1. On SHost01, run the **scsetup** GUI:

   ```
   /usr/cluster/bin/scsetup
   ```

2. On the Main Menu, select **2) Resource groups**.
3. On the Resource Group Menu, select **3) Add a data service resource to a resource group**.
4. On the Add a Data Service Resource to a Resource Group panel, enter `yes` when prompted to continue. The following steps, 5 to 22 on page 543, are all performed on this panel.
5. On the Add a Data Service Resource to a Resource Group panel, enter the additional failover resource group, `websphere_tm-rg`.
6. Select **IBM.WASME WASME server for Sun Cluster** when prompted for the type of resource that you want to add. This is the resource type that was created in “Creating a Node Agent package” on page 521.
7. Still on the same panel, answer `yes` to the question whether the software for this service is installed on each node.
8. Answer `yes` to the question if it is okay to register this resource type now. The panel generates and runs the following command to add the resource type:

   ```
   scrgadm -a -t IBM.WASME
   ```

9. Press Enter to continue.

10. When prompted for the name of the resource that you want to add, enter a name for the resource. Our resource name is `wasmem09ME-rs`.

11. When you see the section that is shown in Example 13-35, type `yes` to override the default setting.

Example 13-35 Adding an application server port for monitoring

```
This data service uses the "Port_list" property. The default
    "Port_list" for this data service is as follows:

        <NULL>

    Please check the documentation for this resource type for more
    information on how the list should be set for this resource.

    Do you want to override the default (yes/no) [no]?
```

12. Fill in an application server port.

    The probe monitoring system from Sun Cluster tries to connect to this port to determine whether the application server is running. We use the application server's SIB_ENDPOINT_ADDRESS.

    To find the application server's SIB_ENDPOINT_ADDRESS port, open the WebSphere Administrative Console. Select **Servers** → **Applications servers** → **wasmember09** → **Ports**. Open **SIB_ENDPOINT_ADDRESS**.

    The port number for our application server is 7279. Thus, for Port number (Ctrl+D to finish), enter `7279`.

13. Enter `yes` to the question whether this is a TCP port.

14. Press Ctrl+D to stop entering ports.

15. The panel now confirms the TCP port just entered. Answer `yes` if it is correct.

16. Press Enter to continue.

17. When prompted for extension properties that you would like to set, enter `no`. However, if you know that you do want to change properties, you can enter

yes and change properties. The list of properties might look similar to that shown in Example 13-36.

Example 13-36 Extension properties that can be updated

```
Here are the extension properties for this resource:

        Property Name             Default Setting
        =============             ===============

        Confdir_list              <NULL>
        Monitor_retry_count       4
        Monitor_retry_interval    2
        Probe_timeout             30
        Child_mon_level           -1

    Please enter the list of properties you want to set:
      (Type Ctrl-D to finish  OR  "?" for help)
```

18. Answer yes when prompted to proceed with the update if you are ready to add the application server resource.

19. Based on your entries, the Add a Data Service Resource to a Resource Group panel generates and runs this command:

    ```
    scrgadm -a -j wasmem09ME-rs -g websphere_tm-rg -t IBM.WASTM -y
    Scalable=false -y Port_list=7279/tcp
    ```

    If something goes wrong with adding the resource, you can copy and reuse this command instead of stepping through **scsetup** again.

20. Press Enter to continue.

21. Answer no when prompted to enable this resource.

22. Enter q twice to exit **scsetup**

23. The next step is to update the Thorough_Probe_Interval and Retry_Interval properties.

    We update these to allow the application server enough time to start before Sun Cluster checks to see if it is running. If Sun Cluster probes too soon, it tries to stop and restart the application server in the middle of its initial start process. The Retry_Interval must be equal to or greater than the Thorough_Probe_Interval multiplied by the Retry_Count. To update these properties:

    a. Update the Retry_Interval to 360:

       ```
       scrgadm -c -j wasmem09-rs -y Retry_Interval=360
       ```

    b. Update the Thorough_Probe_Interval:

       ```
       scrgadm -c -j wasmem09-rs -y Thorough_Probe_Interval=180
       ```

These values assume the Retry_Count is set to 2.

24. Activate the resource group using the **scswitch** command:

```
scswitch -e -j wasmem09ME-rs
```

Review the application server's log for a successful start. If there are problems starting, you can stop the resource using the **-n** parameter on **scswitch**:

```
scswtich -n -j wasmem09ME-rs
```

25. Use the **scstat** command to review the status of the resources. The wasmem09-rs resource should be added to the previously selected resource group under the section Resource Groups and Resources, as shown in Example 13-37.

Example 13-37 Resources including the wasmem09ME-rs resource

```
-- Resource Groups and Resources --

            Group Name          Resources
            ----------          ---------
 Resources: db2_db2inst1_0-rg   suncluster db2_db2inst1_0-rs s-suncluster
 Resources: websphere_active-rg nodeagent-rs wasmem09-rs
 Resources: websphere_me-rg wasmem09ME-rs
 Resources: cluster1TM          -
 Resources: cluster2TM          -
```

Sun Cluster now controls the messaging engine, wascluster01.005-wascluster01, and is able to stop and start the messaging engine.

Adding the second cluster member

Repeat these steps to create a package, resource type, and resource for wasmember10.

13.5.7 Testing: failing the Node Agent and application servers

To verify the scalable resource properties and that the Transaction Manager and messaging engine will restart on the second system, reboot the primary machine. The partially started server on the backup machine should finish starting the server with the Transaction Manager and messaging engine.

Rebooting the machine

To test that the Node Agent is highly available on both machines using a failover configuration with Sun Cluster, reboot the machine that acts as the primary. Do the following:

1. On SHost01, use the **scstat** command to check the location of the resource groups. If it is active on SHost02, use the **scswitch** command to swap to SHost01:

   ```
   scswitch -Z -g db2_db2inst1_0-rg
   scswitch -S -g websphere_active-rg
   scswitch -Z -g websphere_tm-rg
   ```

2. Verify that the Node Agents and application servers are started. Open the Administrative Console, and check their status. If they are not running, start them using the **scswitch** command. When starting the resources with the **scswitch** command, watch the application server logs to determine when to start the wasmem09TM-rs and wasmem09ME-rs. Start the wasmem09TM-rs when the application server pauses, waiting for the Transaction Manager to start. Enable wasmem09ME-rs after the server starts.

   ```
   scswitch -e -j nodeagent-rs
   scswitch -e -j wasmem09-rs
   scswitch -e -j wasmem09TM-rs
   scswitch -e -j wasmem09ME-rs
   ```

3. Run Trade 6 to verify that it is working using wasmember09's WebContainer Inbound Chain.
4. Reboot SHost01.
5. Review the application server log on SHOST02, it should complete the start process and possible recover transactions.
6. Run Trade 6 again against the WebContainer Inbound Chain for wasmember09.

13.5.8 Troubleshooting

If you have problems where Sun Cluster attempts to start, stop, and restart the application server or Node Agent repeatedly, you might need to adjust some of the time related properties on the resource. Sun Cluster might be trying to probe the resource too soon. Try adjusting the Thorough_Probe_Interval and Retry_Interval. There are also timeouts on the stop and start commands within Sun Cluster.

You can also adjust the various scripts if necessary. The script to start the application server, wasctrl-as, has a timeout attached to the **startServer.sh** and **stopServer.sh** commands. You might need to adjust this value to fit your environment.

Also, you can put the messaging engine in a separate resource group from the Transaction Manager to have finer control over the resource behavior.

13.6 Reference

- Sun Cluster software

  http://www.sun.com/clusters

- *Sun Cluster Software Installation* and *Sun Cluster Software Administration* guides

  http://docs.sun.com/app/docs

- *Sun Cluster Data Service for WebSphere MQ Integrator Guide for Solaris OS*

  http://docs.sun.com/app/docs/doc/817-4580

- *IBM DB2 Universal Database and High Availability on Sun Cluster 3.x*

  ftp://ftp.software.ibm.com/software/data/pubs/papers/suncluster.pdf

Part 6

End-to-end high availability

14

Backup and recovery of Network Deployment configuration

This chapter describes two simple and tested procedures to backup and restore the configuration data of a Network Deployment configuration in the case of a node failure that is caused, for example, by a hard disk corruption or similar event. It explains the steps that are required to restore a node to a previously saved configuration state using either a file system backup or the command-line tools **backupConfig** and **restoreConfig**.

Important: These procedures should not be seen as a replacement for established and more powerful backup scenarios for the entire system, using backup tools such as IBM Tivoli Storage Manager or other third-party backup mechanisms. Only WebSphere Application Server configuration data is restored. Other data that is required typically for a production cell, such as the SSL keyring, is not backed up or restored by the WebSphere supplied utilities.

14.1 Network Deployment configurations

In a typical Network Deployment cluster configuration, capacity and reliability can be increased by horizontal scaling, for example by using multiple physical nodes inside a cell and application clustering across these nodes (see Figure 14-1). The Deployment Manager holds the master configuration repository and provides a centralized point of cell administration.

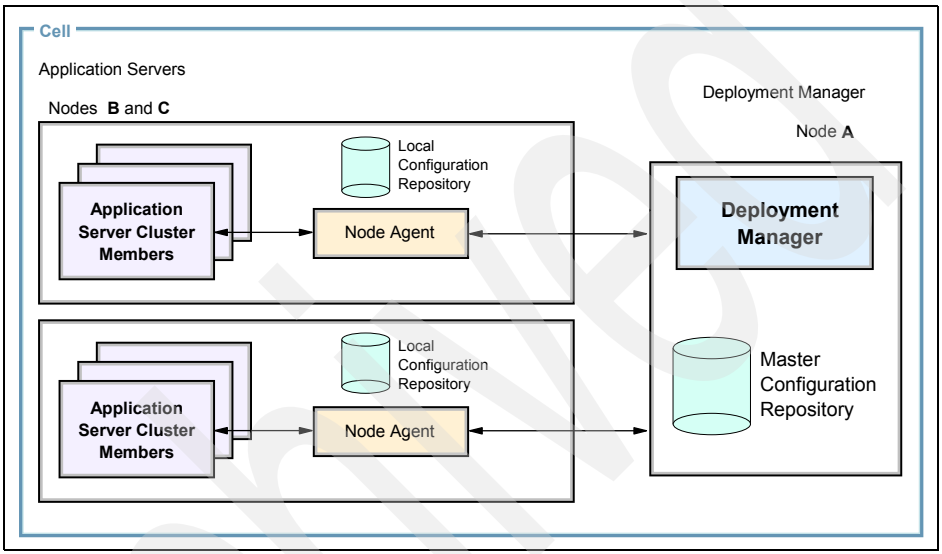


Figure 14-1 A Network Deployment cell configuration

In the case of an application server node failure, the other cluster members take over the load of the failed one transparently, and after restoration, the node continues to participate in sharing the workload. If the Deployment Manager node fails, the application servers in the cell keep working. However, making configuration changes without the central administration view becomes inefficient and burdensome. You can choose to configure your system in a highly available cluster to minimize downtime, but this would require additional hardware and software and adds to the complexity of the system configuration.

For an alternative implementation of Deployment Manager high availability *without clustering*, refer to the developerWorks article by Tom Alcott, which provides additionally hints and insights into system backup and restoration. You can find this article at:

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

14.1.1 Backup methods

There are several different methods to backup and restore a WebSphere Application Server node. One of them is to use the **backupConfig** and **restoreConfig** command-line tools that come with the product. Another is to save the files inside the <*WAS_HOME*> directory tree using backup software such as IBM Tivoli Storage Manager. Example 14-1 shows the output when backing up a repository using the **backupConfig** command.

Example 14-1 Backing up a repository using backupConfig

```
C:\WebSphere\AppServer\bin>backupConfig -nostop
ADMU0116I: Tool information is being logged in file
           C:\WebSphere\AppServer\profiles\dm\logs\backupConfig.log
ADMU0128I: Starting tool with the dm profile
ADMU5001I: Backing up config directory
           C:\WebSphere\AppServer/profiles/dm\config to file
           C:\WebSphere\AppServer\bin\WebSphereConfig_2004-10-28.zip
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.............
ADMU5002I: 411 files successfully backed up

C:\WebSphere\AppServer\bin>dir WebSphereConfig*
 Volume in drive C is WINDOWS2000
 Volume Serial Number is 842D-BF03

 Directory of C:\WebSphere\AppServer\bin

10/28/2004  02:44p           3,786,819 WebSphereConfig_2004-10-28.zip
               1 File(s)      3,786,819 bytes
               0 Dir(s)   1,433,133,568 bytes free
```

Independent of the backup strategy that you implemented, it is recommended to take a backup before major configuration changes or updates. You should also have backups scheduled on a regular basis. On UNIX platforms, schedule backup jobs using the **cron** utility. On Windows platforms, use the Scheduler facility, or use your backup software's proprietary mechanisms to achieve the same.

Tip: The **backupConfig** command uses the **-nostop** option, which allows you to perform a backup in a production environment without stopping any service.

This chapter focuses on the steps that are necessary to restore a failed system in a *Network Deployment configuration*. Restoring a base configuration, that is a stand-alone node, is straightforward and is not discussed here. Refer to *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451 for more information about backing up and restoring a stand-alone node installation.

14.2 Node failure scenarios

We discuss the following two node failure scenarios in a Network Deployment cell configuration:

1. Failure of the Deployment Manager node
2. Failure of a WebSphere Application Server node

14.2.1 Failure of the Deployment Manager node

Because the Deployment Manager provides a centralized view for the administration of the cell, backing up the master cell configuration repository on a regular basis is an important task. If the Deployment Manager fails, the cell's application servers continue to do their work. However, you cannot change the configuration using the Deployment Manager's Administrative Console. It is possible to administer individual application server nodes using **`wsadmin`**, but any local changes are overwritten when the Deployment Manager is online again. In this case, an automatic or manual repository synchronization is done.

You can find additional techniques for improving system availability, such as Deployment Manager high availability through external clustering, in Chapter 9, "Configuring WebSphere Application Server for external clustering software" on page 285.

14.2.2 Failure of a WebSphere Application Server node

A WebSphere Application Server node holds a partial copy of the master cell configuration repository. When it fails, its configuration state is available in the master copy on the Deployment Manager node, making the restoration process very easy.

To check for an unavailable node:

1. Log on to the WebSphere Administrative Console and select **System Administration** → **Node agents**. The Node Agent on the failed node should be unavailable because the Deployment Manager has lost contact (see Figure 14-2 on page 553).

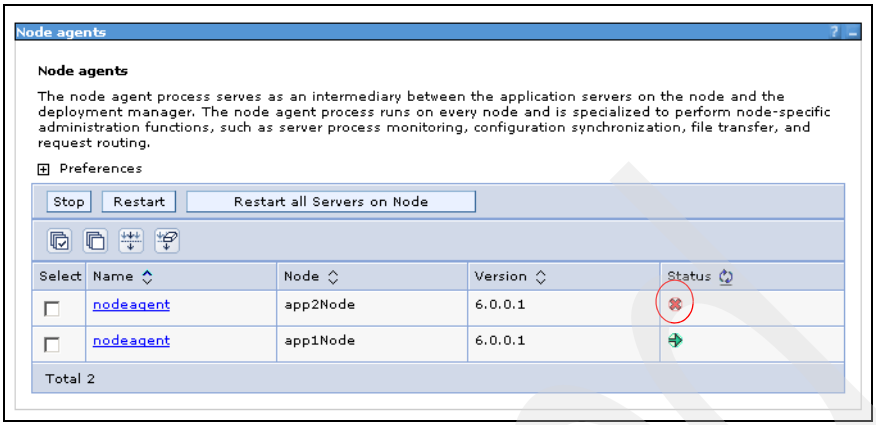


Figure 14-2 Status of failed Node Agent (unavailable)

2. Select **System Administration** → **Nodes**. Figure 14-3 shows that the status of the failed node is *Unknown*.

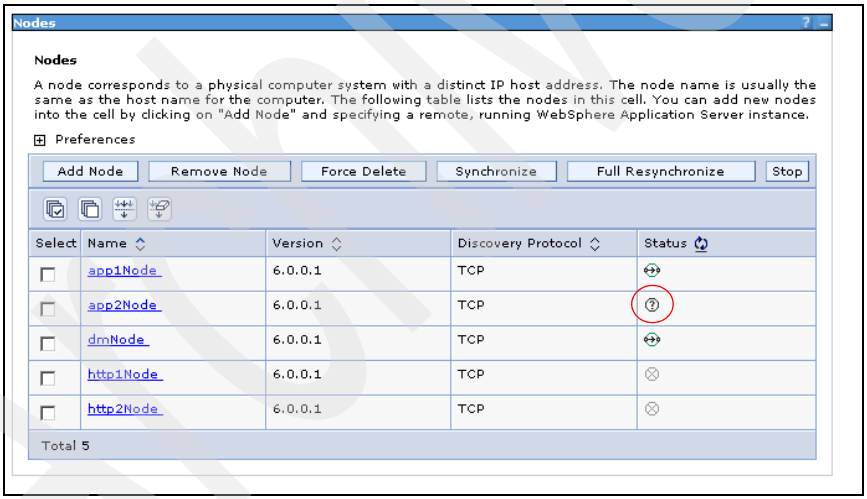


Figure 14-3 Display node status for failed node

14.3 Node recovery

This section explains the steps that are necessary for recovering a failed node, either the Deployment Manager or an application server node, using two different methods. Which method you use depends on individual requirements and general conditions of your environment (for example, backup storage cost,

budget, software licenses for backup software, mean time to repair, service level agreements, and so forth). These recovery methods are as follows:

- Using a conventional file system backup.
- Using the **backupConfig** and **restoreConfig** command-line tools.

Note: The recommended method for system recovery is a file system based backup and recovery because the **backupConfig** method does not address SSL keyring files and other property files.

We assume that the failed node has been either repaired or replaced up to the point where the WebSphere Application Server prerequisites are met again (that is, hardware requirements are met and the operating system and all required OS fixes are installed). See the WebSphere Application Server hardware and software requirements at:

http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html

Regardless of the recovery method, always start with a re-install of WebSphere Application Server so that the system registry is set up correctly.

Tip: Consider installing the software using response files to avoid errors during the (repeated) product and fixes installation and for a quick and silent (unattended) setup process. For detailed information about silent product installation, refer to the InfoCenter, search for *silent install*.

14.3.1 Recovery using file system backup and restore methods

The advantages of using a file system backup and restore over the **backupConfig** method are that fixes and fixpacks, and the entire configuration including the federated state of the node are restored instantly, shortening node repair time. Although we could restore the file system backup only, we decided to initially install the WebSphere Application Server software so that all necessary entries and information in the system registry are set. A disadvantage is that more files than necessary are backed up, increasing the storage space requirements on your archiving system.

Prerequisites

For the complete recovery of a failed node, the following prerequisites have to be met:

- The installation files for WebSphere Application Server Network Deployment are available.
- An up-to-date file system backup of the *<WAS_HOME>* subdirectory of the failed node, which is part of a Network Deployment cell, is available.

Restoring the Deployment Manager node

To restore the Deployment Manager node:

1. Reinstall WebSphere Application Server Network Deployment with the same parameters or features that were used at the time of the initial system setup.

   For information about installing WebSphere Application Server Network Deployment and IBM HTTP Server, refer to *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451 or to the InfoCenter.

2. If the Deployment Manager process is running after installation, stop it using the **stopManager** command.

3. Restore the file system backup of the *<WAS_HOME>* directory.

Important: Remove (or rename) the *<WAS_HOME>* directory before restoring the backup. Otherwise this would result in a garbled repository, containing data of the initial default and the newly restored configuration.

4. Start the Deployment Manager using the **startManager** command.

   Check the status of the nodes using the Administrative Console. You should find all previously defined nodes, Node Agents, application servers, enterprise applications, and so forth.

5. Optionally update the cell configuration with the latest changes since the backup snapshot was taken. Save and synchronize the configuration. Any changes made to the individual application servers are lost because they are overwritten by the master configuration data from the Deployment Manage node.

Restoring a WebSphere Application Server node

Because all configuration data is kept in the master repository on the Deployment Manager node, the following procedure is fairly easy due to the centralized configuration replication mechanism in WebSphere Application Server. To restore the node:

1. Reinstall WebSphere Application Server Network Deployment (and IBM HTTP Server if collocated on the same system) with the same parameters or features that were used at the time of the initial system setup.

   You can retrieve the node name of the failed node from the WebSphere Administrative Console by expanding System Administration and selecting **Nodes** (as shown in Figure 14-3 on page 553).

   For information about installing WebSphere Application Server and IBM HTTP Server, refer to *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451 or to the InfoCenter.

2. Restore the file system backup into the *<WAS_HOME>* directory.

Important: Remove (or rename) the *<WAS_HOME>* directory before restoring the backup. Otherwise, the backup results in a repository that contains data of the initial, default, and the newly restored configuration.

3. Perform a full node synchronization using the **syncNode** command. The Node Agent contacts the Deployment Manager and performs a full repository synchronization. When it terminates, the node has the latest configuration data.

   For detailed instructions on how to use **syncNode**, refer to Chapter 5 in *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451 or to the InfoCenter.

4. Start the Node Agent using the **startNode** command.
5. Start the application server or cluster member on the newly restored node.

14.3.2 Recovery using backupConfig and restoreConfig

The **backupConfig** and **restoreConfig** command line utilities are part of WebSphere Application Server and provide an easy way of backup and recovery of the configuration repository in case of a node outage.

The advantage of using these tools is that you do not have to create a full file system backup. One drawback is that you have to reinstall all previously installed fixes and fixpacks manually (in contrary to a file system restore). However, you can speed up this task by using semi-automated, silent installations that use response files.

Another disadvantage is that the utilities do not back up SSL keyring files (default location <WAS_HOME>/etc) and the <WAS_HOME>/property directory. These files contain additional security related information and are not saved! Therefore, you should only use this method if a file system based recovery approach is not practicable or if you are able to use the default, initially installed versions of these files.

For detailed instructions on how to use **backupConfig** and **restoreConfig**, refer to Chapter 5 in the *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451 or to the InfoCenter.

Prerequisites

For the complete recovery of a failed node, the following prerequisites have to be fulfilled:

- The installation files for WebSphere Application Server Network Deployment are available.
- All previously installed fixes and fixpacks are available.
- A previously saved configuration backup file as produced by the **backupConfig** command is available. The default filename of a backup archive is <WAS_HOME>/bin/WebSphereConfig_yyyy-mm-dd.zip.
- Additionally, some properties files are not restored, because they are located outside the <WAS_HOME>/config directory subtree. In a cluster configuration these files are identical to the ones on the other cluster members in the cell. Depending on whether you adapted these files, you either have to repeat that configuration step or to copy them manually from another cluster member or the Deployment Manager to the restored node.

  These files are as follows (not necessarily a complete list):

  - Property files, for example, security related files in the <WAS_HOME>/properties/ directory:
    - sas.client.properties
    - sas.server.properties
    - wsadmin.properties
  - Files in the <WAS_HOME>/etc/ directory (for example, SSL keyring files such as plugin-key.kdb).
  - The plug-in configuration file, if it was modified manually on the Web server node (or nodes).

Restoring the Deployment Manager node

To restore the Deployment Manager node:

1. Reinstall WebSphere Application Server Network Deployment with the same parameters/features as used at the time of the initial system setup. Specify the same values for Node name, Host name, and Cell name in the setup dialog that were used during the previous installation.

   For information about installing WebSphere Application Server Network Deployment, refer to *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451 or to the InfoCenter.

2. If the Deployment Manager process is running after installation, stop it using the **stopManager** command.

   Reinstall all fixes and fixpacks that were previously installed on the system using the Update Installer product. The Updateinstaller comes with fixpacks,

but you can also obtain it separately from the WebSphere support page at the following URL by selecting the appropriate fixpack in the **Self Help** → **Download** section:

http://www.ibm.com/software/webservers/appserv/was/support/

Refer to the included readme file and documentation on how to use the Update Installer program.

Attention: The Deployment Manager must have the highest fix level in the entire cell!

3. Restore a previously taken configuration backup using the **restoreConfig** command (see Example 14-2). The initial cell configuration data is backed up automatically and replaced by the restored configuration in this step.

Example 14-2 Restoring a configuration using restoreConfig

```
C:\WebSphere\AppServer\bin>restoreConfig WebSphereConfig_2004-10-28.zip
ADMU0116I: Tool information is being logged in file
           C:\WebSphere\AppServer\profiles\dm\logs\restoreConfig.log
ADMU0128I: Starting tool with the dm profile
ADMU0505I: Servers found in configuration:
ADMU0506I: Server name: dmgr
ADMU2010I: Stopping all server processes for node dmNode
ADMU7702I: Because dmgr is registered to run as a Windows Service, the request
           to stop this server will be completed by stopping the associated
           Windows Service.
ADMU5502I: The directory C:\WebSphere\AppServer\profiles\dm\config already
           exists; renaming to C:\WebSphere\AppServer\profiles\dm\config.old
ADMU5504I: Restore location successfully renamed
ADMU5505I: Restoring file WebSphereConfig_2004-10-28.zip to location
           C:\WebSphere\AppServer\profiles\dm\config
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.............
ADMU5506I: 411 files successfully restored
ADMU6001I: Begin App Preparation -
ADMU6009I: Processing complete.
```

4. Start the Deployment Manager using the **startManager** command.

   Check the status of the nodes using the Administrative Console. You should find all previously defined nodes, Node Agents, application servers, enterprise applications, and so forth.

5. Optionally, update the cell configuration with the latest changes since the backup snapshot was taken. Save and synchronize the configuration. Any changes made to the individual application servers is lost because they are overwritten by the master configuration data from the Deployment Manager node.

Restoring a WebSphere Application Server node

To restore a WebSphere Application Server node:

1. Reinstall WebSphere Application Server Network Deployment with the same parameters/features as used at the time of the initial system setup. Specify the same values for *Node name* and *Host Name* in the setup dialog as during the previous installation.

   The node name of the failed node can be retrieved from the WebSphere Administrative Console by expanding **System Administration** and selecting **Nodes** (see Figure 14-3 on page 553).

   For information about installing WebSphere Application Server Network Deployment, refer to *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451, or to the InfoCenter.

2. Reinstall all fixes and fixpacks that were previously installed on the system using the Update Installer product. The Updateinstaller comes with fixpacks, but you can also obtain it separately from the WebSphere support page at the following URL by selecting the appropriate fixpack in the **Self Help** → **Download** section:

   `http://www.ibm.com/software/webservers/appserv/was/support/`

   Refer to the included readme file and documentation on how to use the Update Installer program.

3. Restore a previously taken configuration backup using the **`restoreConfig`** command (see Example 14-2 on page 558). The initial configuration data is backed up automatically and replaced by the restored configuration in this step.

4. Perform a full node synchronization using the **`syncNode`** command. The Node Agent contacts the Deployment Manager and performs a full repository synchronization. When it terminates, the node has the latest configuration data.

   For detailed instructions on how to use syncNode, refer to Chapter 5 in *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451, or to the InfoCenter.

5. Start the Node Agent using the **`startNode`** command.

6. Start the application server or cluster member on the newly restored node.

14.4 Conclusion

Using either file system backups or taking configuration snapshots by running the **backupConfig** command on a regular basis lets you easily restore a failed node in a Network Deployment configuration. If downtime is critical, stick to the file system recovery method. For a small cell, when the focus is centered more around the budget and additional files such as SSL keyring files or property files are no issue, the command-line tools **backupConfig** and **restoreConfig** provide another, cheaper but nearly equally effective alternative.

14.5 Reference material

These documents contain additional information about backing up and restoring WebSphere Application Server:

- Section 5.9, “Managing your configuration files” in the *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.
- Chapter 6, “Administration with scripting” in *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.
- *Implementing a Highly Available Infrastructure for WebSphere Application Server Network Deployment, Version 5.0 without Clustering*, which is available at:

  http://www.ibm.com/developerworks/websphere/library/techarticles/0304_alcott/alcott.html
- White paper *Server Clusters for High Availability in WebSphere Application Server Network Deployment Edition 5.0*, which is available at:

  http://www.ibm.com/support/docview.wss?uid=swg27002473

15

WebSphere end-to-end high availability

A complete WebSphere system includes several components. WebSphere system availability is determined by the weakest part in the chain. This chapter provides an overview of the high availability options that you can use at various layers in building a highly available WebSphere system.

15.1 Introduction

An end-to-end WebSphere environment, as shown in Figure 15-1, involves many different components, such as the Web server, firewalls, application servers, directory servers, databases, and so forth.

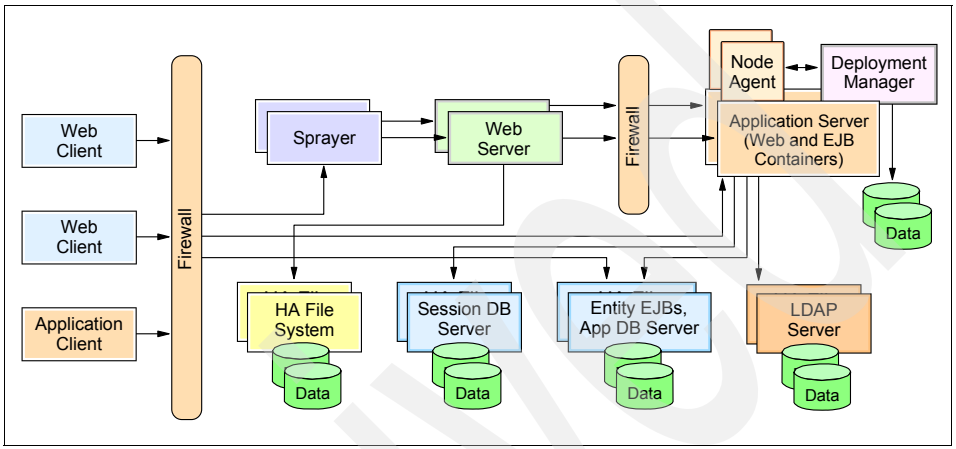


Figure 15-1 End-to-end WebSphere environment

Note: A component box in Figure 15-1 does not mean that this is also a separate hardware system. Many of these components can be collocated on one physical system (even though for security reasons you might want to use multiple LPARs or multiple systems).

Each of these components can become a single point of failure if it is not made highly available, with different effects on your entire system. For example, if your network sprayer, which is the entry point into your system, fails, no Internet clients can access your application any more. If your directory server (LDAP) fails, only those parts of the application that need authentication are affected. However, this could be the most important part of your application so it might still be very important to have a HA directory server. Refer to Chapter 1, “Understanding high availability concepts” on page 3 for conceptual information about WebSphere system availability.

Each component in a WebSphere system offers different options on how it can be made highly available. In this chapter, we discuss HA options for the following:

- WebSphere Load Balancer
- Web server
- Database server
- LDAP Server
- Firewall

This chapter does not discuss HA options for the WebSphere components themselves. For this information, see Chapter 2, “WebSphere Application Server failover and recovery” on page 35.

15.2 WebSphere Load Balancer

In this section, we summarize the capabilities of the WebSphere Load Balancer, which is part of the IBM WebSphere Application Server Network Deployment V6 Edge Components, and provide an overview of how to set up a highly available Load Balancer pair.

The Load Balancer has several components that can be used separately or together to provide failover and high availability support:

- Dispatcher
- Content Based Routing (CBR)
- Site Selector
- Cisco CSS Controller
- Nortel Alteon Controller

We are mainly interested in the Dispatcher component which can be used to distribute workload between multiple Web servers (that is, a Web server cluster). This provides both, scalability and high availability for the Web servers. The workload is dispatched to the Web servers in the cluster based on a load balancing mechanism usually known as IP spraying, which intercepts the HTTP requests and redirects them to the appropriate machine in the cluster, based on weights and 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. See “Web server” on page 565 for additional information.

Being the entry point into your WebSphere system, it is extremely important that your Load Balancer is highly available. This can be achieved by setting up a second Load Balancer server with exactly the same configuration. You have to define one or more heartbeat connections between these servers. The heartbeat mechanism is built into the WebSphere Load Balancer. You have two options to configure such a highly available Load Balancer environment:

- Active/Active configuration with mutual takeover
- Active/Passive configuration

For details on Load Balancer configuration, see Chapter 4 and Chapter 5 of the *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392, where step-by-step configuration instructions are provided.

Figure 15-2 shows an Active/Passive configuration before a failover to the backup LB system.

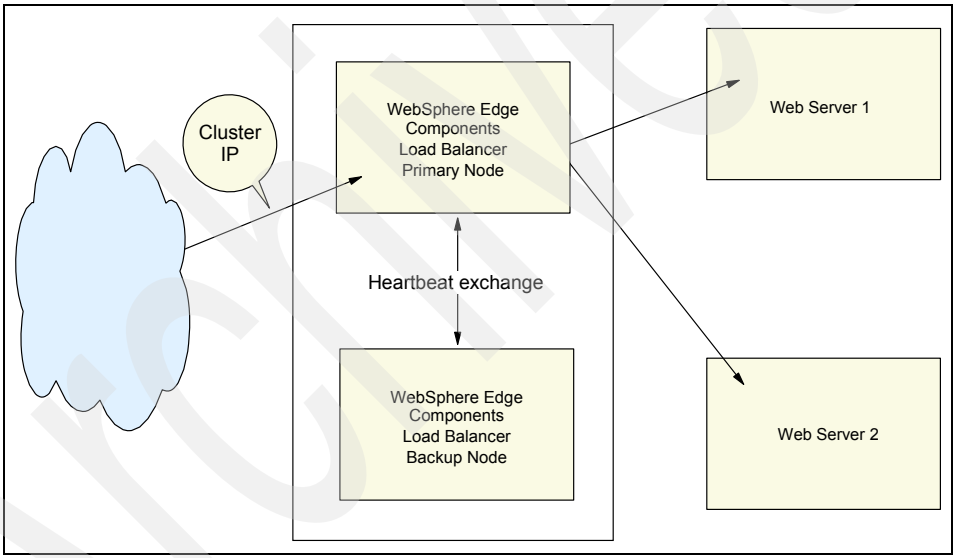


Figure 15-2 Load Balancer configuration

Figure 15-3 shows the takeover of the cluster IP address by the backup Load Balancer server in case of a failure of the primary LB server.

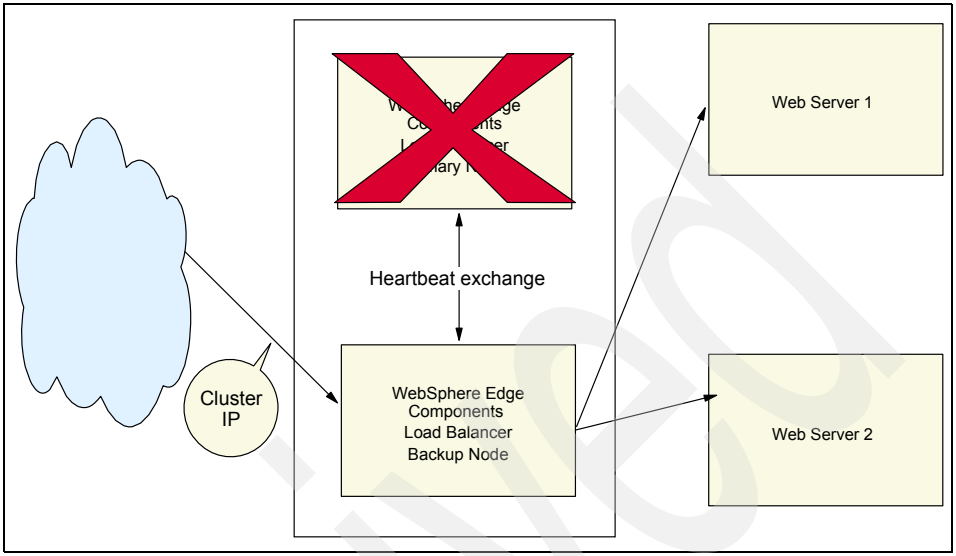


Figure 15-3 Load Balancer configuration after failover

The Load Balancer can also be used to provide high availability for other components, such as firewalls and directory servers. This is covered in the various components' sections in this chapter.

15.3 Web server

A mechanism of providing request distribution and failover for incoming HTTP requests is required. Without this mechanism, the Web server becomes a single point of failure, and scalability is limited to the size of the hardware used to host the Web server. A likely solution is to employ an IP sprayer, such as the Dispatcher component of the WebSphere Edge Components' Load Balancer.

Figure 15-2 on page 564 shows a basic configuration that implements the Load Balancer as the network sprayer (actually the Dispatcher component of Load Balancer is distributing the workload). The Load Balancer is configured in a cluster with a backup Load Balancer, which maintains a heartbeat with the primary server. If the primary system fails, the backup system takes over the Virtual IP Address and processes requests from the HTTP clients.

Each Web server in the topology is configured with at least one physical IP address and a loopback alias configured with a shared Virtual IP Address, also

called the cluster address. HTTP clients make HTTP requests to this Virtual IP Address. All requests are routed to the Load Balancer, which in turn sprays them among the members of the Web server cluster. The Web server cluster consists of identical Web servers running on different physical machines (or in different LPARs). In the event of a failure of one of the Web servers, the Dispatcher discontinues directing work to the failed server.

Note: When i5/OS HTTP servers are part of a Load Balancer Web server cluster, then the following is an example of the command that you must run on each of the i5/OS HTTP server systems to create a Virtual IP Address that can then be started:

```
ADDTCPIFC INTNETADR('10.10.10.100') LIND(*VIRTUALIP)
SUBNETMASK('255.255.255.0')
```

In this command, 10.10.10.100 is the cluster address as defined in the Dispatcher configuration.

Also, you need to ensure that at least two system network interfaces — the cluster address and the interface defined within the Load Balancer that corresponds to the address of that specific HTTP server — are bound to each of the i5/OS HTTP server instances. Of course, choosing to bind all system interfaces to the i5/OS HTTP server instances is another option if that might be appropriate for your environment.

You can use the Manager component of Dispatcher to determine the status of Web servers. It provides weight values of each balanced Web server. Running the Manager component is optional, but it is necessary for dynamic weighting of the Web servers and also for identifying failed servers.

15.3.1 Server affinity

Server affinity allows load balancing for those applications that need to preserve state across distinct connections from a client. If server affinity is not enabled in the Load Balancer, the Dispatcher routes the traffic according to server weights. There are several ways for the Dispatcher to maintain server affinity:

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

The easiest way to configure the server affinity is 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 - until the timeout expires for binding the source IP address and server port. See Chapter 4, “Introduction to WebSphere Edge Components” in *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392 for more information about these options.

Depending on your WebSphere environment you should or should not enable server affinity at the Load Balancer level. Some possible scenarios and their impact on server affinity at the Load Balancer are:

- One Load Balancer with multiple Web servers behind it. Multiple application servers behind the Web servers and every Web server is connected to every application server (See Scenario 1).
- One Load Balancer, multiple Web servers and multiple application servers. Each Web server however is connected to only some of the application servers, not all of them (See Scenario 2).
- Multiple Load Balancers, multiple Web servers and multiple application servers. Every Web server is connected to every application server. The workload between the two or more Load Balancers is distributed via DNS round-robin (See Scenario 3).
- Multiple Load Balancers, multiple Web servers and multiple application servers. This time, however, every Web server is only connected to a certain number of application servers (See Scenario 4).

Note: All of the Load Balancers in these scenarios should be configured with a backup Load Balancer. However, for simplicity reasons, we do not discuss the backup Load Balancer here.

Now what happens with sessions in each of these scenarios? Because you already have session affinity in the Web server plug-in, how should server affinity be configured in the Load Balancer?

Scenario 1

This scenario consists of one Load Balancer and multiple Web servers behind it. Multiple application servers are behind the Web servers and every Web server is connected to every application server. This environment is shown in Figure 15-4.

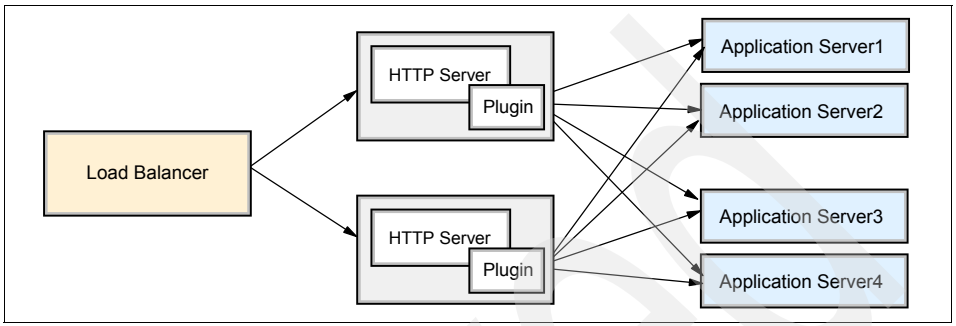


Figure 15-4 Load Balancer server affinity, Scenario 1

For this environment, you do not need to enable server affinity in the Load Balancer. Having it disabled is even the best solution from a performance point of view because the Web server plug-in takes care of session affinity to the application servers (for example based on the JSESSIONID cookie) and all Web servers can handle the incoming requests. Even if a client is directed to a different Web server as the initial request, the plug-in would do it's job and make sure the request is sent to the same application server. So sending the request to the same Web server again is not necessary.

Scenario 2

This scenario consists of one Load Balancer, multiple Web servers, and multiple application servers. However, each Web server is only connected to some of the application servers, not all of them, as shown in Figure 15-5.

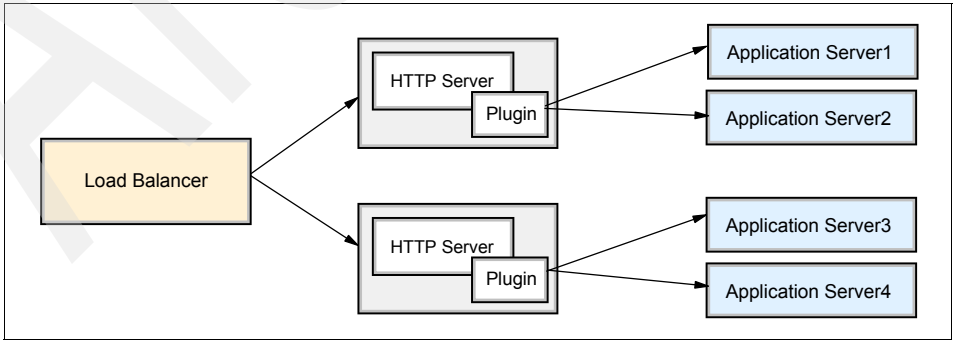


Figure 15-5 Load Balancer server affinity , Scenario 2

For this environment, you *must* enable server affinity in the Load Balancer to make sure the requests are sent to the same Web server again. This is because that Web server is only connected to a part of the application servers in the environment and the plug-in can only distribute to the application servers it knows. S, o you must make sure that subsequent requests are sent to the same Web server.

Restriction: For MAC forwarding, you can only use the Stickiness to source IP address (and cross port affinity) type server affinity. However, this affinity strategy has some drawbacks:

- Some ISPs use proxies that collapse many client connections into a small number of source IP addresses. Thus a large number of users who are not part of the session will be connected to the same Web server. The plug-in still takes care of distributing to the correct application server but you might have more requests going to one of your Web servers.
- Other proxies use a pool of user IP addresses chosen at random, even for connections from the same user, invalidating the affinity.

Scenario 3

This scenario consists of multiple Load Balancers, multiple Web servers, and multiple application servers. Every Web server is connected to every application server. The workload between the two or more Load Balancers is distributed via DNS round-robin. There are two possible configuration options here:

- All Load Balancers are connected to all Web servers.
- Each Load Balancer knows only a part of the Web servers.

This environment is shown in Figure 15-6 on page 570.

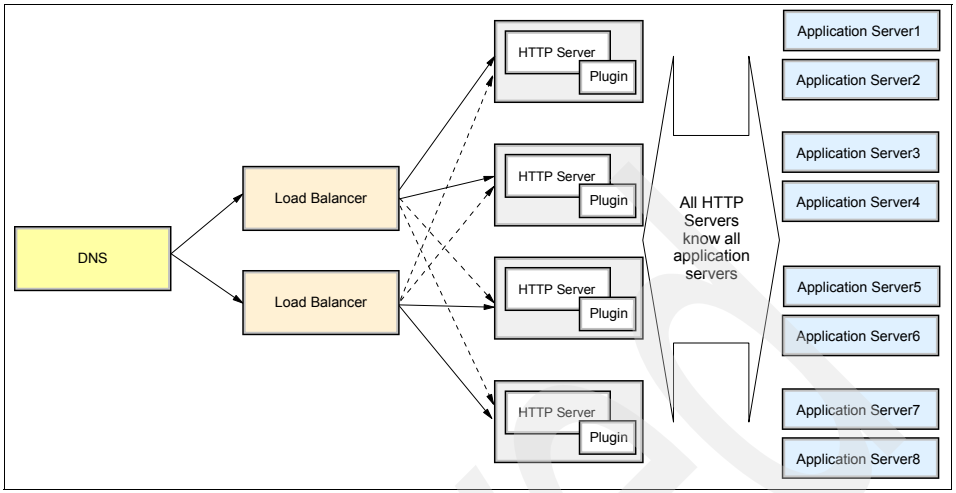


Figure 15-6 Load Balancer server affinity, Scenario 3

You should switch off server affinity in the Load Balancer for both of these configuration options. This is because a subsequent client request might be send to a different Load Balancer, although this is not very likely because, most probably, the IP address for the Load Balancer that is sent back by the DNS will be cached for a sufficiently long time. Thus, subsequent client requests always end up at the same Load Balancer.

For the configuration where all Load Balancers are connected to all Web servers: This scenario is basically identical to Scenario 1 from a workload distribution point of view. When the request arrives at any of the Load Balancers, it can be distributed to any Web server and the plug-in takes care of sending the request to the correct application server. So, for this configuration you could enable server affinity at the Load Balancer. However, for performance reasons, it is best not to do so.

For the configuration where the Load Balancers are only connected to a part of the Web servers, it is important that server affinity is switched off. In the case where a subsequent client request is sent to another LB, that LB would not know the Web server for which the server affinity was previously established. As however all Web servers know all application servers, any Web server plug-in can forward the request to the correct application server based on the previously established session affinity.

Scenario 4

This scenario consists of multiple Load Balancers, multiple Web servers, and multiple application servers. Each Load Balancer knows only some of the Web servers. Also, every Web server is only connected to a certain number of application servers. This environment is shown in Figure 15-7.

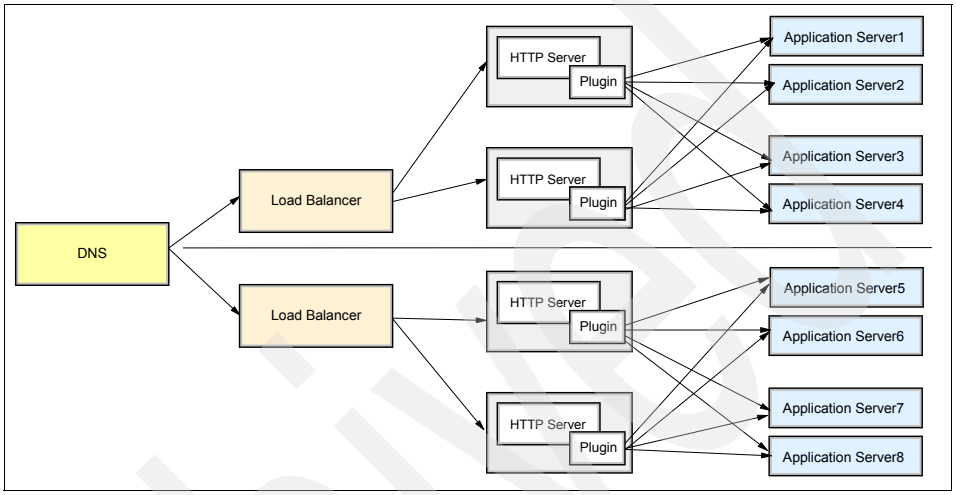


Figure 15-7 Load Balancer server affinity, Scenario 4

For this scenario it is important that a subsequent client request is always sent to the same Load Balancer. This can be achieved by DNS Time-To-Live to enforce the affinity to the Load Balancer or by using different URLs for the Load Balancers and not using DNS round-robin to distribute the workload between the Load Balancers.

Otherwise if a request ends up at the wrong Load Balancer and thus the request is sent to a Web server that does not know the application server that handled the previous request, session information is lost and the client has to start over again.

For server affinity in the Load Balancer: You can choose either setting, on or off, because in the end, it is basically similar to scenario 1. What must be made sure for this scenario is that the request is sent to the right LB.

15.3.2 Web server plug-in file (plugin-cfg.xml) management

Since WebSphere Application Server V6, Web servers are an important conceptual part of a cell configuration. This integration allows you to perform many management tasks for the Web server directly from the Administrative

Console. See Chapter 3 and Chapter 6 of *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392 for details.

One of the big advantages of this new design is that you can (and should) map applications to a Web server during deployment. Consequently, individual plug-in files for each Web server are generated which contain only the entries that are relevant for this Web server. For example, only the applications it is supposed to serve and only the application servers it can forward requests to are listed in this individual plug-in file.

The plug-in properties configuration (accessible through **Servers** → **Web servers** → **WebServerName** → **Plug-in properties**) has two important options that are related to the generation and propagation of the plug-in file:

- Automatically generate the plug-in configuration file
- Automatically propagate plug-in configuration file

Automatic generation of the plug-in file

For this setting to work, the Web server plug-in configuration service must be enabled by selecting **System Administration** → **Deployment manager** → **Administration Services** → **Web server plug-in configuration service** and selecting **Enable automated Web server configuration processing**. It is enabled by default.

Using the automatic plug-in generation option ensures that a plug-in file is automatically updated every time a configuration change happened that is relevant for that plug-in file. The individual plug-in files are generated and stored on the Deployment Manager machine at this location:

<*WAS_HOME*>/profiles/*dmProfileName*/config/cells/*cellName*/nodes/*nodeName*/servers/*serverName*

For example, in our environment with two Web servers on two different nodes in the same cell, the locations are as follows:

<WAS_HOME>/profiles/***itsoprofile01***/config/cells/***wascell01***/nodes/***httpnode1***/servers/***http1***

<WAS_HOME>/profiles/***itsoprofile01***/config/cells/***wascell01***/nodes/***httpnode2***/servers/***http2***

Propagation of the plug-in file to the Web server

It depends on your environment whether you can automatically propagate the plug-in file. For example, if your Web server is on a managed node, then automatic propagation is supported and uses the node synchronization function. If you are using the IBM HTTP Server and a firewall is located between the Deployment Manager and the Web server machine, then the propagation fails

unless you open the correct port in the firewall. You need to open the IBM HTTP Server administrative port, by default this is port 8008. Or, if your Web server is on an unmanaged node and is not an IBM HTTP Server V6, then you are not able to propagate the file automatically.

Using the automatic propagation option ensures that each Web server uses the correct plug-in file and that it is always up-to-date. If your environment does not allow to propagate the plug-in files, then you can copy them manually to the Web servers.

In an HA environment with multiple Web servers, it is very important to copy the correct plug-in file to each one of the Web servers as now it is not a *generic* file any more that contains entries for all application servers and all applications.

You must specify the plug-in installation location during configuration of a Web server in the cell. 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 Linux, this is:

/opt/IBM/WebSphere/Plugins/config/***http1***

If you copy the configuration file or files manually, they must be copied to that very same location.

Manually editing the plug-in file

In WebSphere V6, almost all plug-in related settings can be configured using the Administrative Console. If you have to or prefer to edit the plugin-cfg.xml file directly, you should first disable automatic propagation of the plug-in file to make sure your edited plug-in file is not overwritten by an automatically propagated plug-in file.

15.3.3 Data availability

How the data needed by the Web servers is made available depends upon how (where) the members of the cluster are placed. When all of the Web server machines are located in close geographic proximity then a mirrored or RAID protected, distributed file system can be accessed by all of the members of the cluster. When the Web servers are located around the world then some other mechanism to replicate the data must be employed.

15.4 Database server

A highly available database is a key requirement for a highly available WebSphere solution. Enhancing database availability is critical for WebSphere and can be achieved utilizing software and hardware cluster solutions. However, simply making the database engine highly available does not necessarily create a HA solution. It is important to note that even in a WebSphere environment which employs a HA database there is still an interruption in service while a database is switched from a failed server to an available server. This section focuses on the options available to build a highly available DB2 cluster and the application code (client) implications within WebSphere for database high availability and covers:

- Continuous availability
- Failover availability
- Client application code considerations

The information found in this section is based on the developerWorks article *An Overview of High Availability and Disaster Recovery for DB2 UDB* by Monty Wright, which is available at:

http://www.ibm.com/developerworks/db2/library/techarticle/0304wright/0304wright.html

15.4.1 Continuous availability

Continuous availability demands that the database engine be available for processing SQL transactions 100% of the time. This type of availability is typically only implemented in the most critical of applications. To achieve this goal, total redundancy is required. That means you must have two systems that are fully independent of one another, both in hardware and software.

Essentially, SQL transactions take place on both systems. The failure of one system will cause no interruption of transaction processing on its partner. To make this a reality the application must be aware of both systems and implement each transaction as a *distributed unit of work* (DUOW) across both systems. A DUOW is series of SQL statements executed as a single transaction that is coordinated between systems. The application will submit a transaction to both systems and receive return codes from both systems upon success or failure. The application can then continue to process another DUOW or perform another type of operation. If a failure takes place during which one database system can no longer function, then the application, which is coded to catch the error, can continue to process its workload with the remaining system with no interruption.

To implement a DUOW requires a type 2 database connection and a transaction monitor. A type 2 database connection establishes an environment for a DUOW.

A transaction monitor is responsible for implementing the DUOW and insuring completion or rollback of transactions in the DUOW. DB2 can act as a transaction monitor or you can use a transaction monitor from another software vendor. Figure 15-8 illustrates such a configuration.

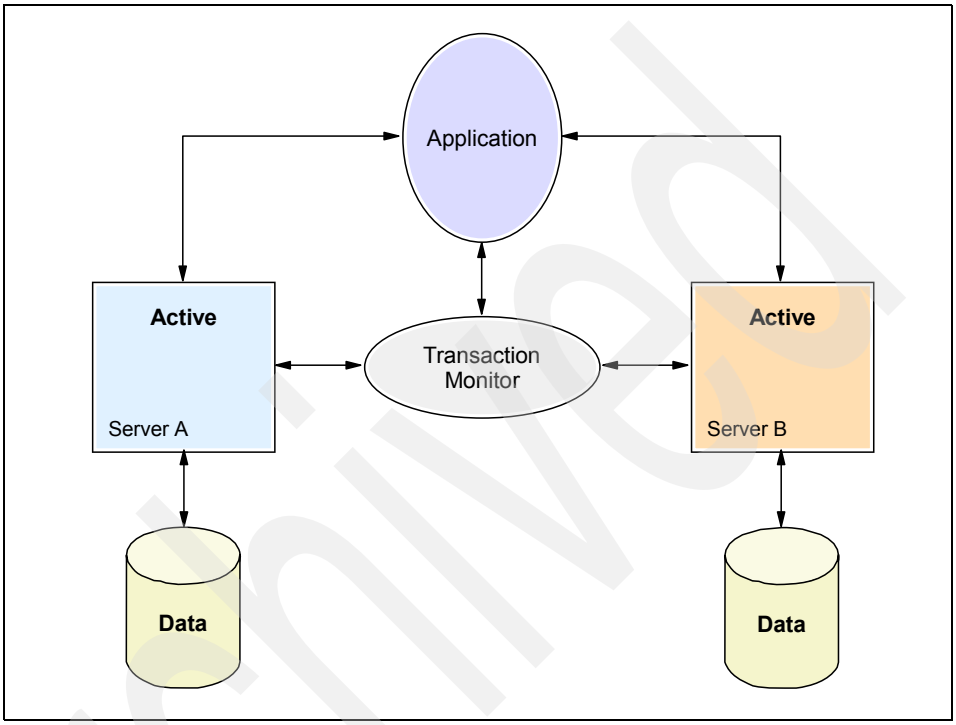


Figure 15-8 Database: continuous availability, DUOW

15.4.2 Failover availability

Failover availability is differentiated from continuous availability by the fact that for some period of time, however small, the database engine is not available for transaction processing. The essential elements for this type of solution are:

- Primary and secondary systems
- Failure detection
- Data source movement

The two systems have copies of the database data, and when a failure is detected, a failover takes place. In the failover process, the data source is moved from the primary to the secondary system.

There are two types of failover availability: *synchronous* and *asynchronous*.

Synchronous availability

Synchronous availability guarantees that the data sources on the primary and secondary systems are identical, and complete continuity is maintained after a failover. Synchronous availability is the most preferred and commonly used approach and involves tight integration of the database software with specialized HA software to produce a HA cluster. HA software support varies by operating system platform. Available HA solutions include:

- High Availability Cluster Multi-Processing (HACMP) for AIX
- Tivoli System Automation (TSA) for AIX and Linux
- Sun Cluster for Sun Solaris
- VERITAS Cluster Server for UNIX, Linux, and Windows
- Microsoft® Cluster Server (MSCS) for Windows
- iSeries clustering with synchronous remote journaling or business partner replication solutions

All these solutions essentially work the same way. If there is a failure, the database server can move from one machine to a backup system. To accomplish this task the HA software moves all the necessary resources to the secondary system. These resources include the disk resources of the physical database, the network resources, and the database server resources.

In the HA cluster solution, a single copy of the physical database is stored on a shared storage system. In the DB2 environment only one system can *own* the storage array at a time. When a failure is detected, ownership of the storage is moved from the primary system to the secondary system. The network resources are moved as well. Finally, the database server resources are started on the secondary system and the database is made available. See Figure 15-9.

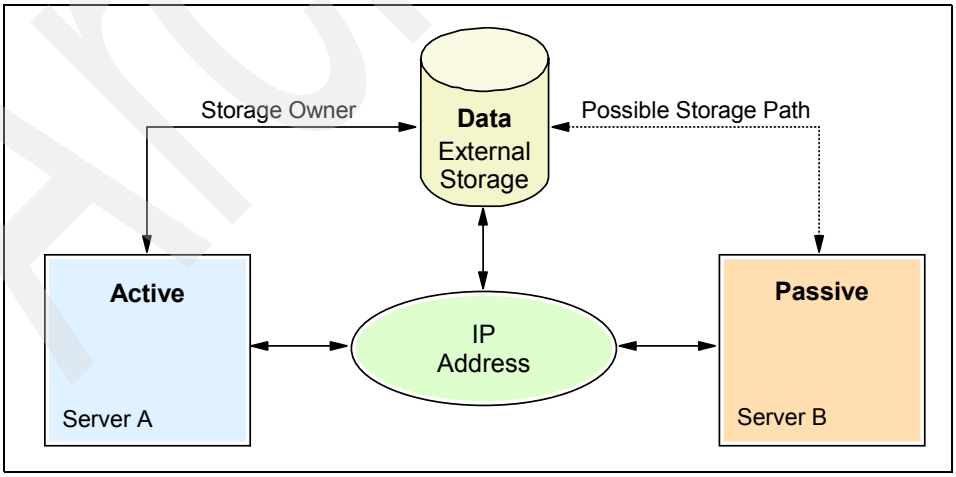


Figure 15-9 Database: failover availability

The detection of a failure is performed by a heartbeat connection between the servers. This heartbeat is a function of the HA software and is aware of both hardware and software failures.

Because there is only a single copy of the database, it is always in sync. The time for the failover and restart of the database engine depends on several factors:

- The time needed to detect the failure.
- The length of time necessary to move database resource dependencies (storage array, networking resources, and so forth).
- The time required for the DB2 engine to perform crash recovery.

DB2 always performs crash recovery when the database is not shut down properly. Crash recovery is the processing of the log files, making sure all committed transactions are written to disk and uncommitted transactions are rolled back. The time required to perform this operation depends upon the amount of *open* work in the database logs at the point of failure. The entire failover could take just a few seconds, or longer if a large workload needs to be processed from the log files.

One advantage of this type of availability solution is that it does not require that any changes be made to the application or to the client configuration directories. The HA software provides a Virtual IP Address resource for database connections. The IP address fails over when a failure is detected, and the same connect statement can be used by the application that was used before. When a failover takes place, all applications are disconnected, and the client returns a communication error condition to the application. When the database server is running on the secondary system, the application can simply reissue the connect statement and continue to process work as before.

This solution is known as a hot standby configuration. However, the secondary system does not have to remain idle while waiting for a failover. The systems can also be configured in a mutual takeover configuration where both servers are actively hosting different databases. Each machine is prepared to take over the workload of its partner in the event of a failure.

See the following sections for information about how to obtain available Cluster Agents and for setup documentation for the various clustering products:

- 11.1.5, “Using DB2 with HACMP” on page 427
- 10.1.5, “Using Cluster Agent for IBM DB2 UDB” on page 373
- 12.1.5, “Using Cluster Agent for IBM DB2 UDB” on page 448
- 13.1.5, “Using the Cluster Agent for DB2” on page 490

Asynchronous availability

Asynchronous availability does not guarantee that the primary and secondary system databases are completely in sync. The method of moving database changes from the primary to the secondary system varies, but the process produces a window of time during which data has not migrated from one system to the other. The amount of data might be very small and the window very short, but it must be taken into consideration when you're deciding on a solution.

The options for implementing an asynchronous availability solution with DB2 include IBM DB2 Replication and High Availability and Disaster Recovery (HADR). IBM DB2 Replication is discussed in *A Practical Guide to DB2 UDB Data Replication V8*, SG24-6828. For information about the DB2 HADR feature refer to:

http://publib.boulder.ibm.com/infocenter/db2help/index.jsp?topic=/com.ibm.db2.udb.doc/admin/c0011267.htm

15.4.3 Client application code considerations

WebSphere allows (client) application code to recover from the interruption and subsequent restoration of a database service, through the use of IBM extensions to the JDBC 2.0 API. These extensions allow clients connecting to or applications running within an application server the capability to reconnect to a database server, if it has either:

- Recovered from a failure
- Recognized that the database is not responding to requests.

The first JDBC extension allows client code to catch a com.ibm.websphere.ce.cm.StaleConnectionException. This exception maps multiple SQL return codes, which indicate a database server failure or outage, to a single exception. Catching this exception allows application code the ability to recognize the need to attempt a reconnection to a database, after service is restored, and is part of the WebSphere programming model for application components (servlets, JSPs, and EJBs).

From an application code perspective stale connections are connections which are no longer usable, for example, if the database server is shut down or the network is experiencing problems. When a StaleConnectionException is detected in the application server runtime, the connection pool is flushed and repopulated. Explicitly catching a StaleConnectionException is not required, because applications already catch java.sql.SQLException and StaleConnectionException extends SQLException. Specifically coding to catch a StaleConnectionException can trigger an application to perform additional (custom) recovery steps. However, in many instances application programmers that develop on WebSphere (normally) need not custom recovery, because there

is an automatic reconnection with the server runtime. The use of StaleConnectionException should be limited to applications that need an extra level of transparent recovery, in addition to that provided by WebSphere. Custom recovery code can be very difficult to develop, especially in complex multiple resources configurations or logic flows.

The second extension allows for a com.ibm.ejs.cm.pool.ConnectionWaitTimeoutException to be caught by the application code. This exception occurs when the connection timeout parameter, for a data source, is exceeded. The timeout parameter specifies how long an application will wait to obtain a connection from the pool, if the maximum number of connections is reached and all of the connections are in use. When an exception is received, an application should resubmit a request to obtain a free connection.

Sample code is provided in Appendix A, “Handling StaleConnectionException” on page 599.

15.5 WebSphere MQ (and other messaging providers)

If you are using any other WebSphere supported external messaging system, such as WebSphere MQ, then you need to make sure that this messaging system is made highly available by using the HA functions provided by the messaging vendor. A combination of those built-in HA functions and the use of clustering software might also be valuable.

WebSphere MQ

For WebSphere MQ on distributed platforms, the approach is to use WebSphere MQ clustering, which means that there are multiple clustered queue managers available. If a queue manager becomes unavailable, new messages are routed around the failed queue manager. However, each of the queue managers owns a particular message queue (a so-called private message queue) and messages in that queue are only accessible as long as the queue manager (QM) is running. Messages that are already in the queue when a QM fails, cannot be accessed before that QM is available again.

Clustering software, such as HACMP or TSA, can be used to restart the queue manager on a backup system and thus make the messages in its private queue available again.

For information about availability of Cluster Agents or setup documentation for the various clustering products, see the following sections:

- 11.1.4, “Using WebSphere MQ SupportPac for HACMP” on page 427
- 10.1.4, “Tivoli System Automation and IBM WebSphere MQ” on page 373
- 12.1.4, “Using Cluster Agent for IBM WebSphere MQ” on page 448
- 13.1.4, “Using the Cluster Agent for WebSphere MQ” on page 490

In addition to WebSphere MQ clustering, WebSphere MQ for z/OS also supports shared message queues that can be accessed by multiple queue managers. If one queue manager fails, another queue manager can access and process the messages.

All of these HA options and scenarios, including limitations and information about when to use which HA option, are covered in detail in the white paper *Understanding high availability with WebSphere MQ* by Mark Hiscock and Simon Gormley, which is available at:

http://www3.software.ibm.com/ibmdl/pub/software/dw/wes/pdf/0505_hiscock_MQ_HA.pdf

Default messaging provider

High availability for the default messaging provider is provided by the HAManager. The HAManager is covered in Chapter 6, “WebSphere HAManager” on page 175. In addition, many details and sample scenarios are described in Chapter 12, “Using and optimizing the default messaging provider” of the *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392.

15.6 LDAP Server

A directory is often described as a database, but it is a specialized database that has characteristics that set it apart from general-purpose relational databases. One special characteristic of directories is that they are accessed (read or searched) much more often than they are updated (written). Lightweight Directory Access Protocol (LDAP) is a popular technology for accessing common directory information. LDAP has been embraced and implemented in most network-oriented middleware. Building LDAP-enabled networks and applications is a common practice in enterprise applications. WebSphere is LDAP-enabled. When the LDAP server fails, WebSphere cannot access directory data, such as security data, and hence fails to service client requests. Therefore, building a HA LDAP is a part of the highly available WebSphere system.

15.6.1 Using clustering software and shared disks

In a way similar to building an HA database, you can build an HA LDAP service with clustering software as shown in Figure 15-10. Two nodes are interconnected with public networks and private networks. Private networks are dedicated to the heartbeat message. A shared disk that is connected to both nodes is used to store common directory data. Clustering software and LDAP software are installed in each node. A resource group is created, and can be failed over from one node to the other under the control of the clustering software. Instead of individual physical host IP addresses, the cluster IP address is used to access the LDAP service.

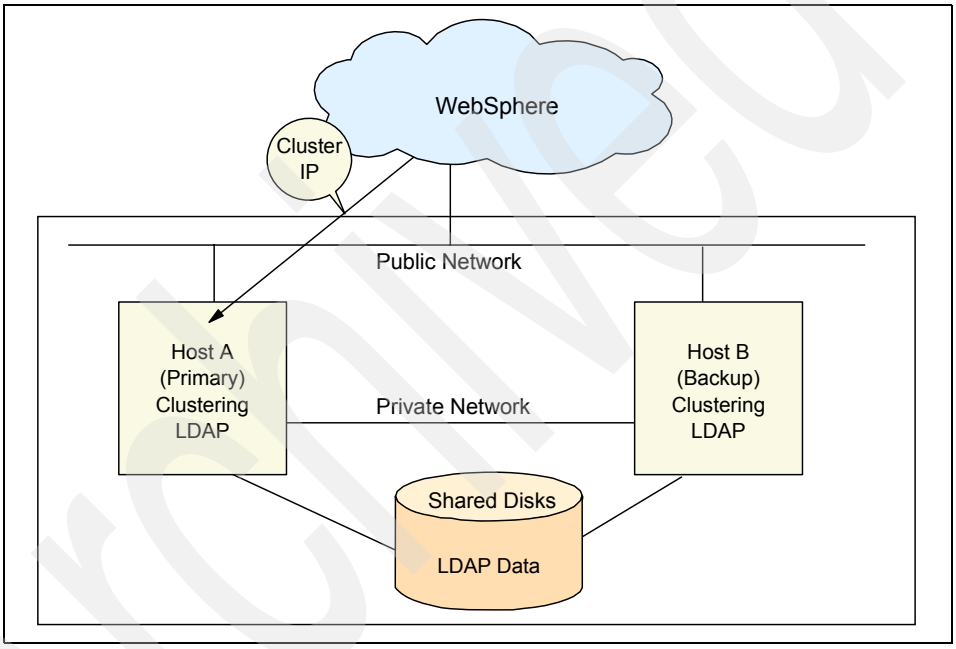


Figure 15-10 Clustered LDAP with shared disks

The cluster IP address is moved to the healthy backup node under the control of the clustering software when the primary node fails and the cluster detects the failure through the heartbeat mechanism. The LDAP client (WebSphere) uses the same IP address (the cluster IP address) to access the LDAP service, as shown in Figure 15-11. You can configure the service to fall back automatically to the primary node when the primary node is up again, or you can do it manually.

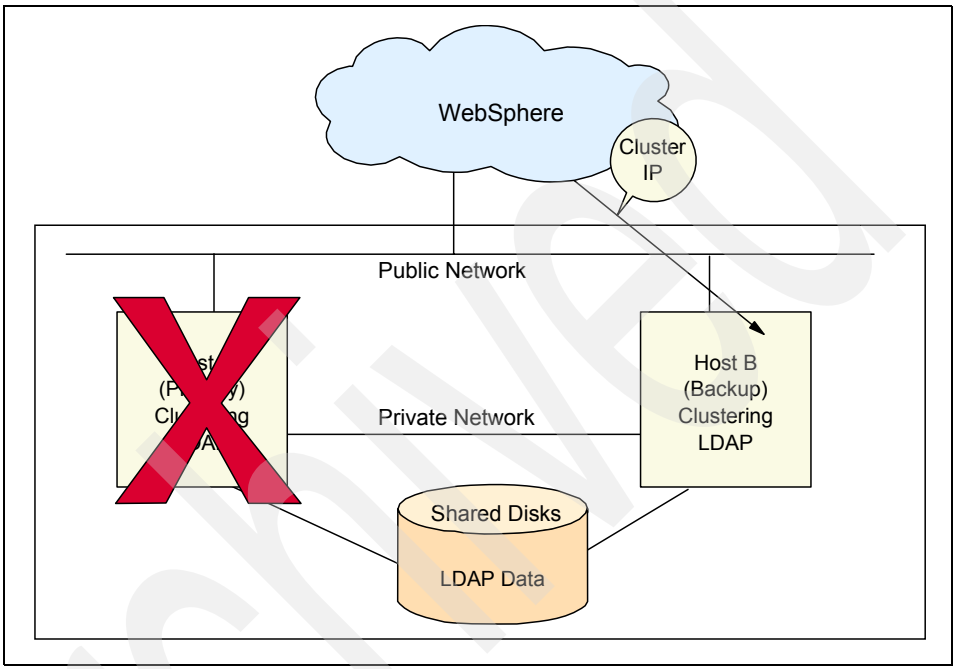


Figure 15-11 Clustered LDAP with shared disks after failover

15.6.2 Using clustering software and LDAP master-replica

The multihost shared disk is used in the above configuration for storing LDAP data. In addition, most LDAP vendors support a master and replica architecture that makes it possible for you to configure a HA LDAP without shared disks. Install clustering software on both nodes, and configure LDAP to use local data.

The primary node is configured as the LDAP master, and the backup node is configured as the LDAP replica, as shown in Figure 15-12. Any LDAP change requests that go to the replica server are referred to the master server because the replica server cannot change data. The master server sends all changes to the replica server to synchronize its data.

Note: Replication is not part of the LDAP standard. Therefore, the master-replica function is implemented vendor specific and can differ from product to product. Refer to your LDAP administration manuals for information about how to use and configure this option.

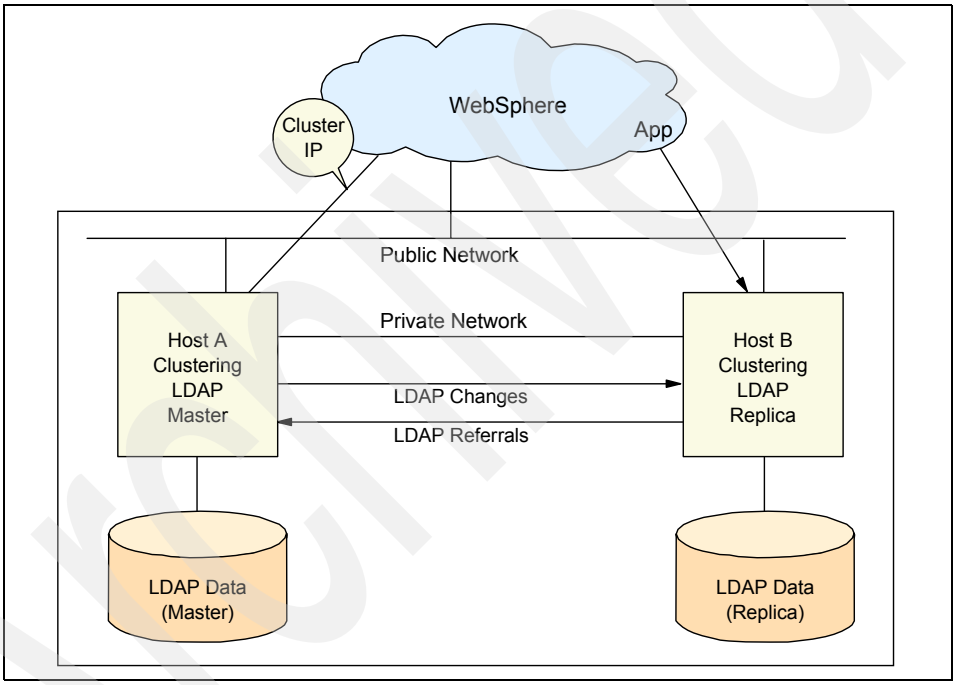


Figure 15-12 Clustered master-replica LDAP with individual disks

When the primary server (master) is down due to some network, hardware, or software reason, the LDAP service is moved to the backup server under the control of the clustering software. The replica server is promoted temporarily to the master server and continues to service LDAP requests, as shown in Figure 15-13.

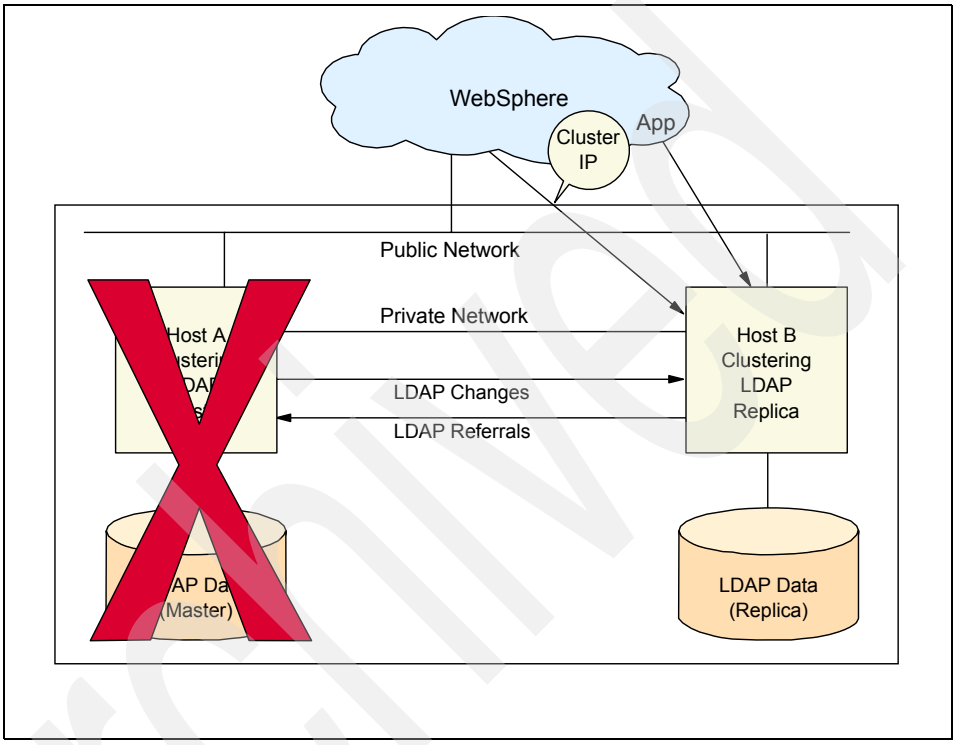


Figure 15-13 Clustered master-replica LDAP with individual disks after failover

When the primary node is up again, you can move the LDAP service back to the primary node. You should not configure automatic fallback, because by doing so, you will lose all updates. You need to export the latest data from the backup server manually and import it to the primary server before you start the primary LDAP server again. It takes time to synchronize the data in the master server in this share-nothing configuration. In the shared disks LDAP configuration, because you use the same data in the shared disks, you do not need to synchronize the data between two servers. However, it is easier to configure the cluster without shared disks.

15.6.3 Using a network sprayer (Load Balancer)

Besides of configuring a HA LDAP with clustering software, you can build a low-cost, easy-to-configure HA LDAP with a network sprayer such as the WebSphere Edge Components' Load Balancer, or a DNS server that has a load balancing function (DNS round-robin), as shown in Figure 15-14.

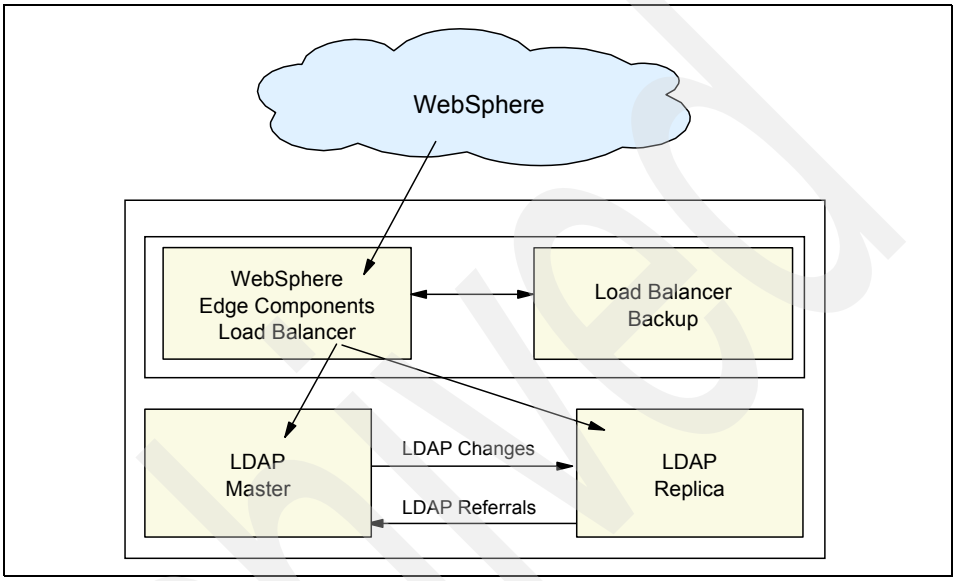


Figure 15-14 LDAP and Load Balancer, master-replica

The Load Balancer distributes client requests to both servers. When one of the LDAP servers fails, the requests are directed to the other server, as shown in Figure 15-16 on page 587.

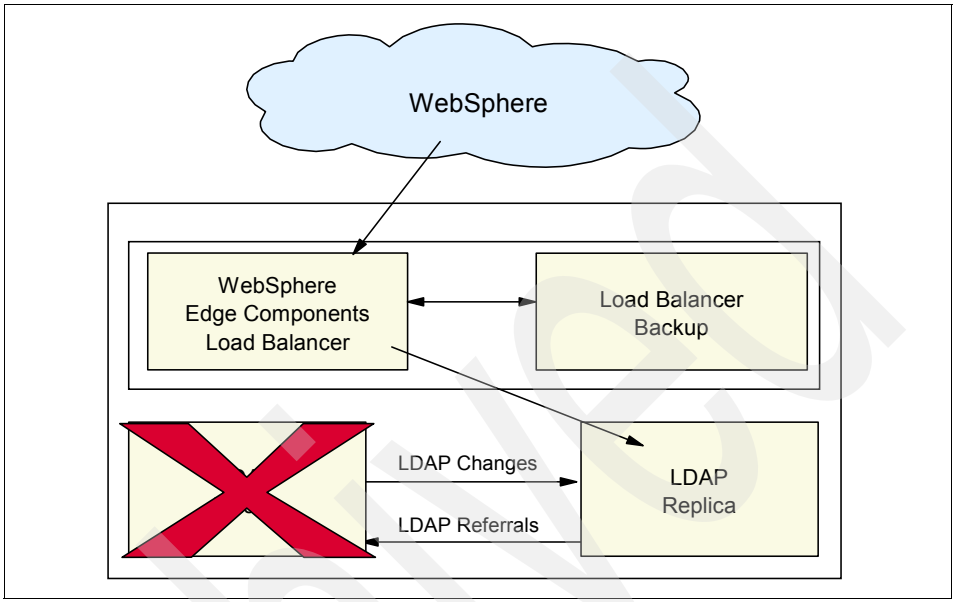


Figure 15-15 LDAP and Load Balancer after failure of an LDAP server

Because the Load Balancer has a backup configured, this system also caters for a failure of the Load Balancer as shown in Figure 15-16.

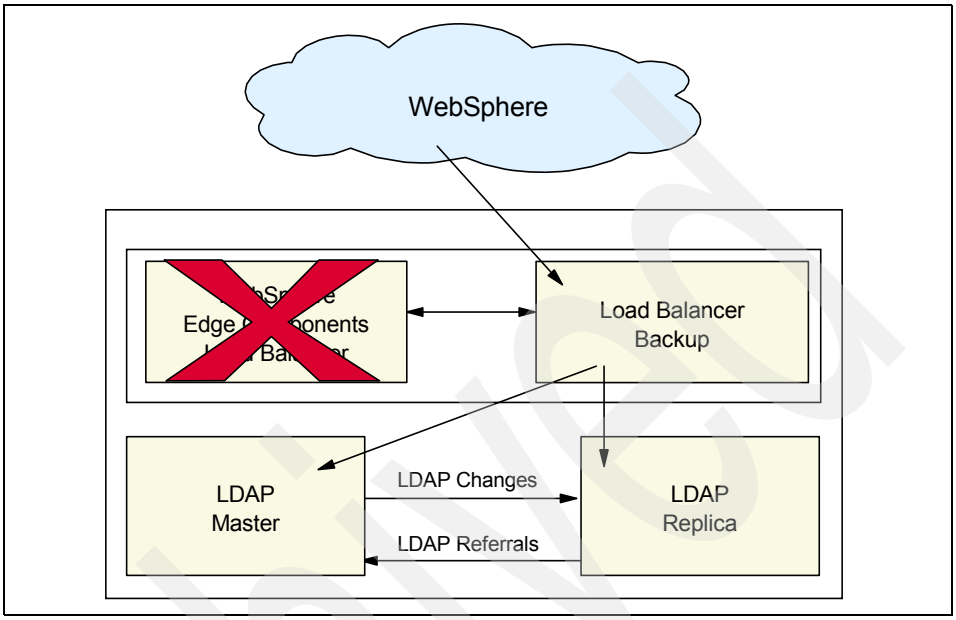


Figure 15-16 LDAP and Load Balancer after failure of the LB server

15.6.4 Using a network sprayer (Load Balancer) with LDAP peer replication (multi-master)

This setup is similar to the previous one with the exception that both LDAP servers are masters. It is possible to have several servers acting as masters for directory information, with each master responsible for updating other master servers and replica servers. This is referred to as *peer replication*. Some vendors also refer to this replication topology as multi-master. See Figure 15-17 on page 588.

Peer replication can improve performance, availability, and reliability. Performance is improved by providing a local server to handle updates in a widely distributed network. Availability and reliability are improved by providing a backup master server ready to take over immediately if the primary master fails. Peer master servers replicate all client updates to the replicas and to the other peer masters, but do not replicate updates received from other master servers.

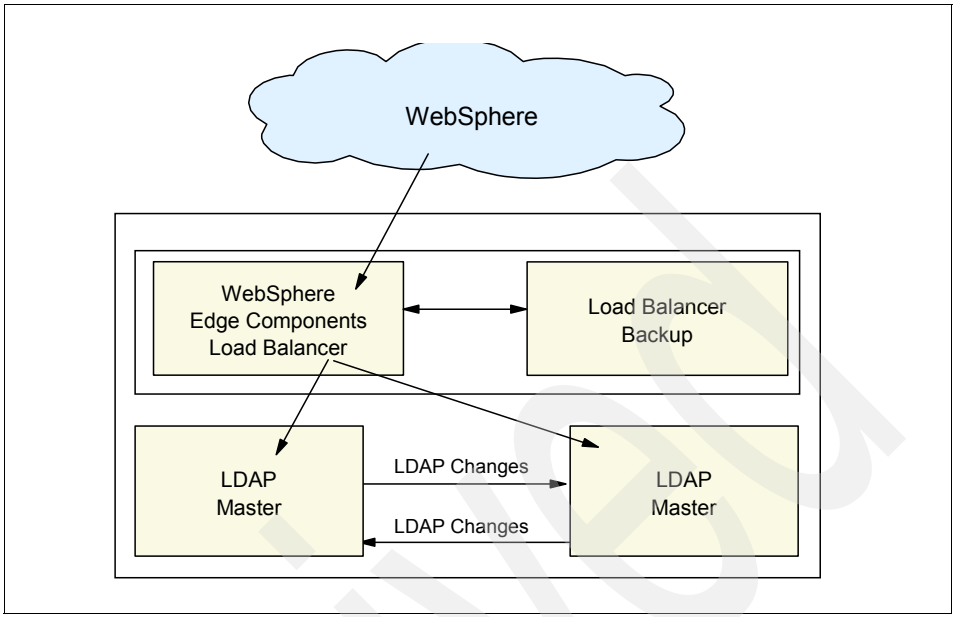


Figure 15-17 LDAP and Load Balancer, multi-master/peer replication

Note: If a high volume of directory changes could occur in a brief interval of time, then you should consider using an "always true" Load Balancer rule and adding just one LDAP server to that rule to ensure that all requests get directed to only one LDAP server while still ensuring that the other LDAP server can service requests in the event of a failure of the designated LDAP server associated with the always true rule.

15.6.5 Conclusions

For high-end enterprise applications, combining clustering software and a network sprayer can improve both, LDAP availability and scalability by reducing downtime and providing more servers, as shown in Figure 15-18 on page 589. During the clustering transition downtime, you can still access LDAP servers (read only) with this configuration. You can also partition your directory structure to enhance scalability, and use approaches discussed here to enhance availability.

Further information about implementing LDAP can be found in the *Understanding LDAP Design and Implementation*, SG24-4986.

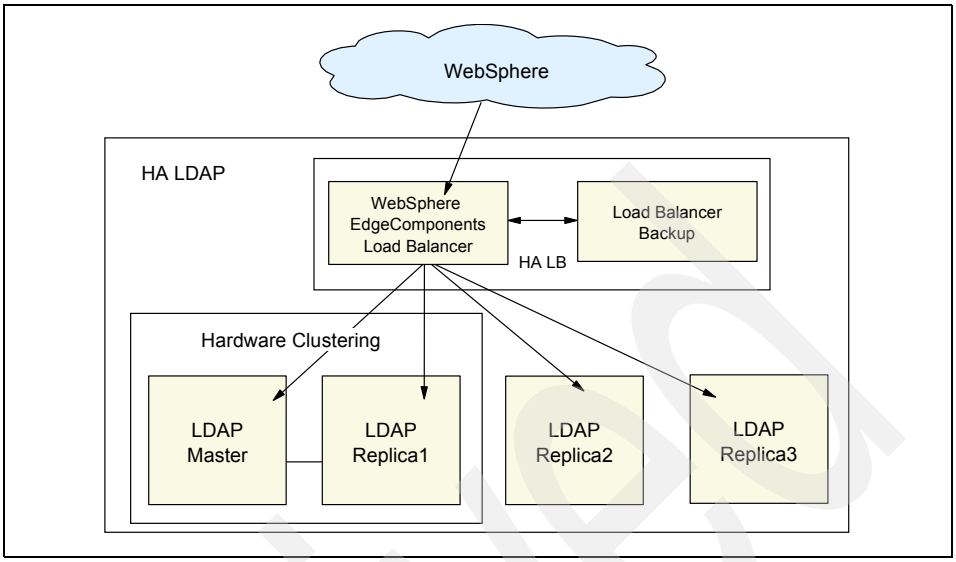


Figure 15-18 Combined LB and clustered LDAP

15.7 Firewall

Usually, a WebSphere production system includes at least one firewall. Two firewalls are commonly used to create a demilitarized zone (DMZ) to enhance WebSphere system security. If the firewall fails, customers are not able to access any services and the site can be exposed to security risks (hacker's attacks). Therefore, the firewall availability is an important part of the WebSphere system's availability.

We can configure a highly available firewall environment by using two separate firewalls on two hosts. Some firewall products provide built-in HA features, such as state synchronization of the firewall modules that allow active connections to continue after failover. However, you also need a synchronization mechanism to synchronize the security policy (filter rules and users) between the firewalls or you will have a a single point of failure.

In this section, we discuss two advanced solutions:

- Building an HA firewall with clustering software such as HACMP, TSA, and so forth.
- Building an HA firewall with a network sprayer such as WebSphere Edge Components' Load Balancer.

15.7.1 Using clustering software

As shown in Figure 15-19, clustering software is used to provide highly available service IP addresses, resource groups, and fault-monitoring mechanisms.

Each node has a complete installation of the firewall software. Configure both nodes in such a way that the equal and interchangeable configurations on both nodes are assured.

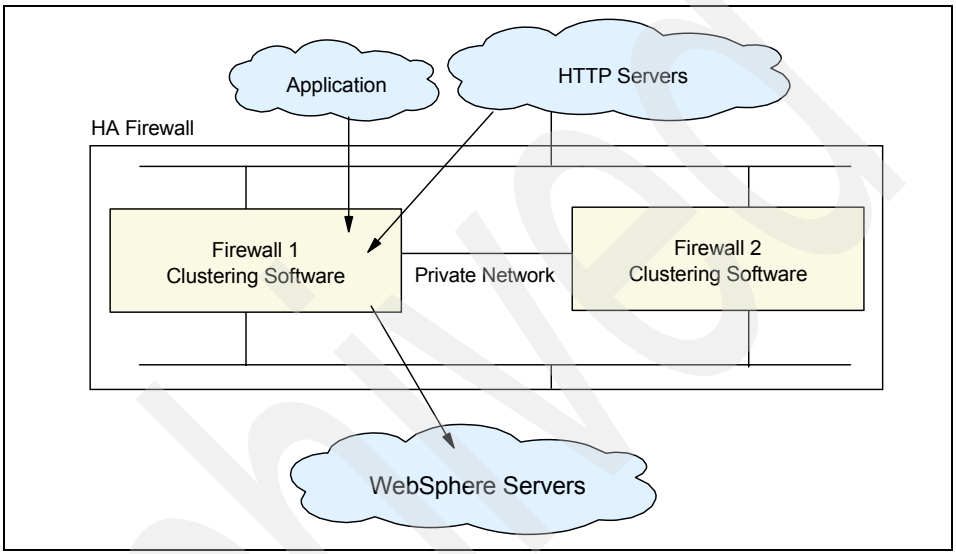


Figure 15-19 Clustered firewall for high availability

When the firewall process, network, or machine itself goes down, the clustering software will detect it and relocate the resource group, including the service IP address, to the backup node, then start the firewall service, as shown in Figure 15-20 on page 591. The highly available firewall environment is reestablished after a failover. As soon as the primary firewall is up, the firewall service can automatically fall back to the primary node, or you can do this manually (this is determined by the clustering configuration settings).

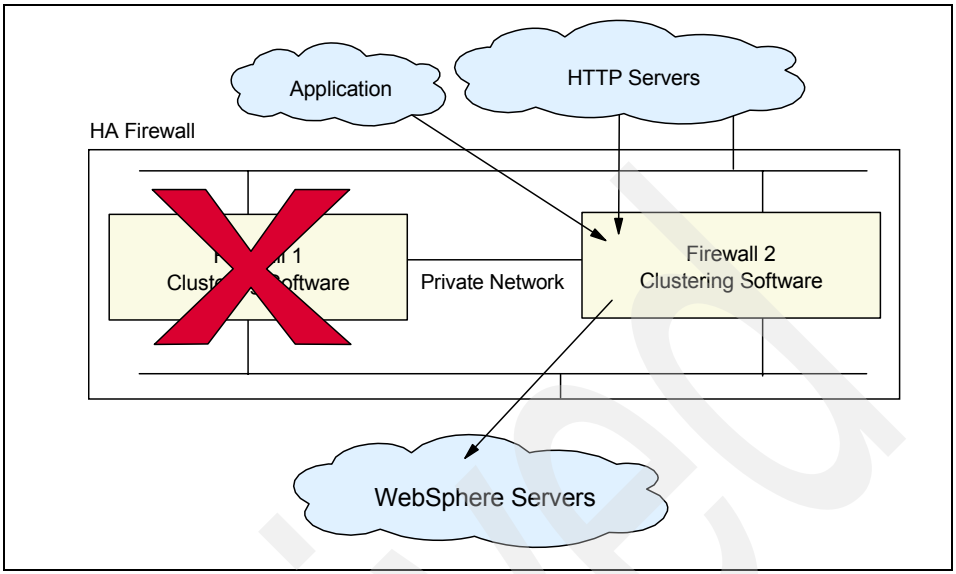


Figure 15-20 Clustered firewall after failover

15.7.2 Using a network sprayer

The HA firewall through clustering service discussed above is rather costly and its configuration is complicated. It can be used by high-end customers. You can also set up an HA firewall with a network sprayer such as WebSphere Edge Components' Load Balancer, as shown in Figure 15-21 on page 592. The Load Balancer is a load balancing product that divides up the workload generated by new connections among a group of back-end servers. This can be done either by changing the assignment between the host name and the IP address (DNS redirection), or by rerouting TCP and UDP traffic directly to the server with the lowest workload. Load Balancer also recognizes server failures and automatically keeps new requests from being dispatched to the failed server.

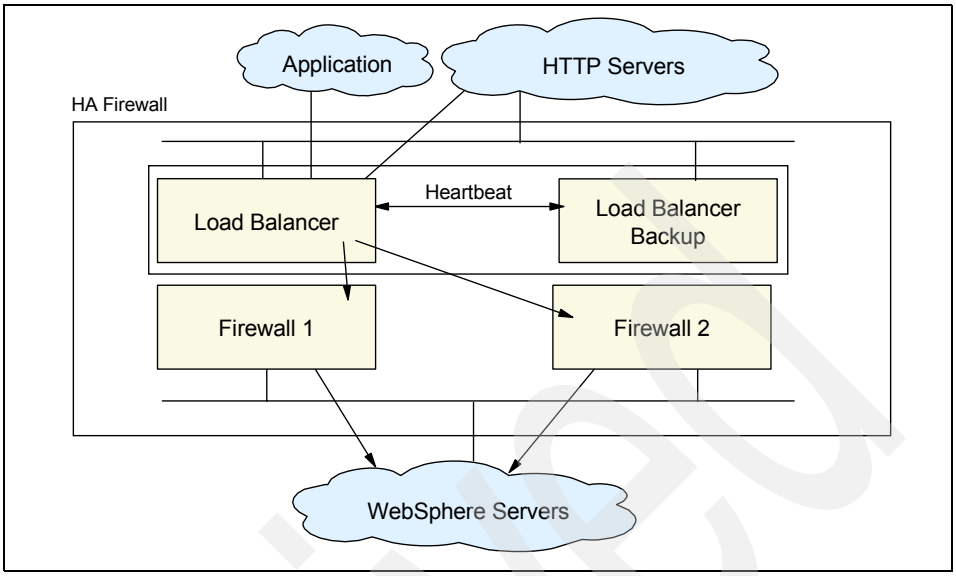


Figure 15-21 Load Balancer configured firewall

As mentioned before, the Load Balancer is a sophisticated load balancing tool. The clients target a cluster IP address configured on the LB server. The requests are then rerouted by the Load Balancer to the server with the lowest workload. The LB recalculates the workload of the servers either with information collected by the LB itself, such as the number of active connections and new connections, or with system information collected by the Metric Server running locally on the servers, such as CPU load or memory utilization.

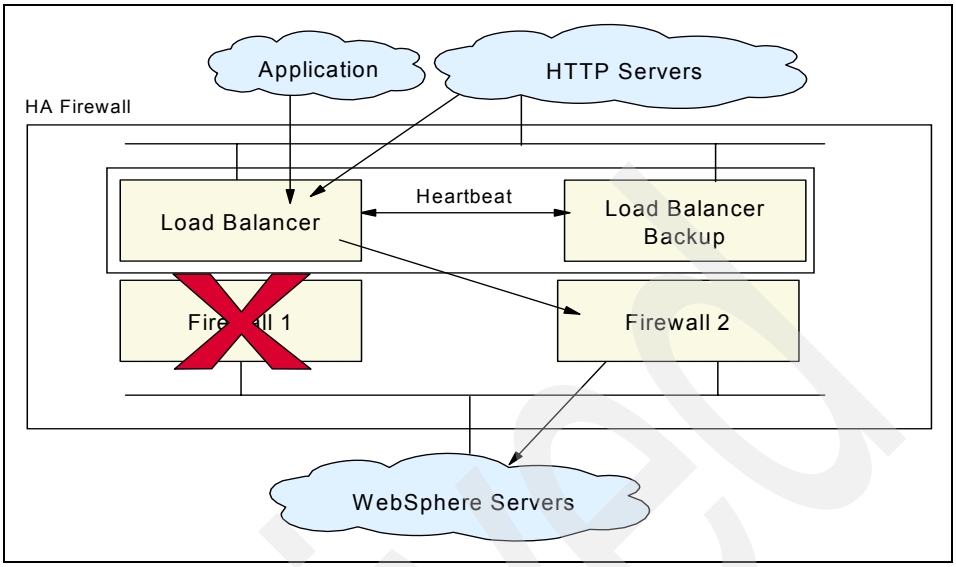


Figure 15-22 Load Balancer configured firewall after failover

The Load Balancer server is a single point of failure in the system. To prevent this, Load Balancer gives us the option to configure a backup Load Balancer server that automatically takes over in case of a failure. The actual load information and client routing tables are shared between the two Load Balancer servers, so nearly all connections can be preserved in the case of a breakdown. Some external scripts are automatically executed during takeover, and they can be modified to provide the high availability of the firewall. This is similar to the scenario described for the LDAP servers with the Load Balancer, see Figure 15-16 on page 587.

Detailed information about the Load Balancer and the WebSphere Edge Components can be found in Chapters 4 and 5 of the *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392.

15.7.3 Conclusions

In making a comparison of the high availability features of clustering and network sprayer, the following aspects should be considered.

Using third party clustering software is complicated and the clustering software needs to establish multiple connections between the two servers. These connections must be allowed by the firewall. Configuration and troubleshooting of this solution is not an easy job and produces an environment that is not easy to understand. Therefore, the firewall administrator must have a good understanding of the clustering software to keep this solution running.

Using the high availability function provided with LB is fairly simple. The Load Balancer executes shell scripts for start and takeover events, which configure the cluster IP addresses either to the loopback device or to the network card, depending on the state of the system. These scripts can be customized for your environment and can include commands to start and stop application proxies and generate alerts for takeover events. Because LB only uses one TCP connection on a dedicated port for the heartbeat, and ping for testing network functionality, there are few changes on the firewall configuration, so setup is fairly easy.

Both software packages can test network connectivity to determine if a network card on the active firewall has failed or if there is a general network failure by pinging other systems on that network.

15.8 Summary

We have discussed various techniques to implement high availability end-to-end WebSphere production systems. Availability commonly refers to *uptime*. Using techniques discussed here, you can build the entire WebSphere system with 99.5% availability or better.

High availability is not only for 7x24 businesses. There are two types of downtimes: planned downtime and unplanned downtimes. The techniques we discussed are for both planned downtime and unplanned downtime. Clustering techniques make rolling upgrades of hardware and software and hot replacement possible. Nobody knows when unplanned downtime will occur. It might occur during your business hours, even though you are in a 5x8 business. Clustering techniques help the system with automatic fault detection and service recovery. If you want to avoid interruptions to your operations due to system failures, then you need to use these techniques whether you are a 7x24, 6x20, or

5x8 business. Non-7x24 business hours provide opportunities for planned downtime for system maintenance and software/hardware upgrades off-line, but your system can still fail at any time due to hardware, software, and network problems.

Any high availability implementation needs high investment in hardware, software, and skilled personnel. Therefore, it is important to evaluate how much you will lose if your system is down during your operation hours.

WebSphere production system availability is determined by the weakest link in the WebSphere system chain. Therefore, it is very important to evaluate each part of the end-to-end WebSphere system high availability and eliminate all single points of failure, as shown in Figure 15-23.

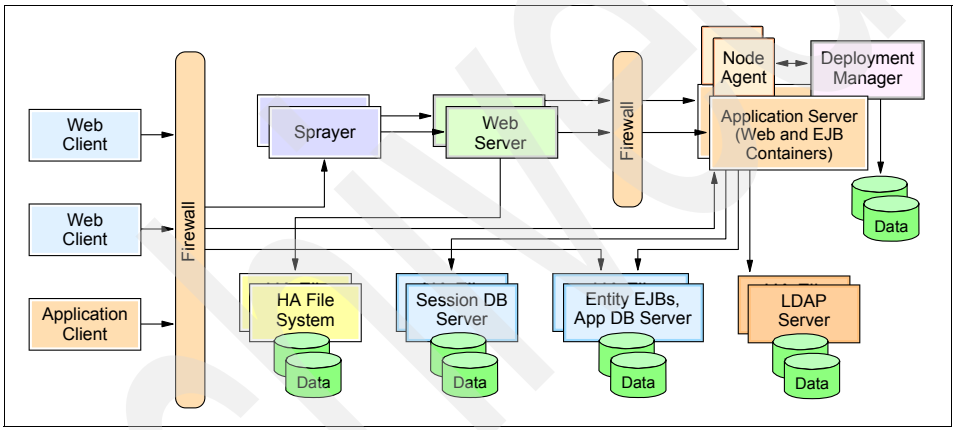


Figure 15-23 An end-to-end WebSphere system that removes single points of failure

15.8.1 Process availability and data availability

Process availability is achieved by multiple processes of an application, such as WebSphere workload management using server clusters, multiple Web servers, multiple database instance processes, multiple firewall processes, and multiple LDAP server processes. Usually, clients find the available process using 1-to-n mapping, redirection (IP spraying), or transparent IP takeover.

Data availability is achieved by replication or clustering. When data is shared by multiple processes, data integrity should be ensured by distributed lock management. Data is either stored in memory or on disk. For in-memory or local data, we need to maintain client request affinity to access the same data. It is very important to make sure that data inconsistencies be corrected before any process uses data, because a failed process can damage data integrity.

Depending on data change and access frequencies, we have different approaches to achieve data high availability:

- Type I. Static data

  There are no changes for a period of months. An example is software install binaries. This static data is usually placed in individual hosts. For convenience of management, it can also be placed in shared disks or file systems.

- Type II. Rarely changing data with planned change time (change period: several hours to several days)

  Examples are firewall configuration files, Web server configuration files, WebSphere configuration files, or HTTP static files. You can copy these files to different nodes (replication). However, an HA file system can help to minimize your administration burden. If, for example, you have 10 Web servers and you need to copy HTTP files to 10 Web servers every day (assuming that you change Web pages once a day), content management software could be used to reduce the administrative work involved in managing this data.

- Type III. Rarely changing data with unplanned change time

  An examples is LDAP data. Clustering or replication can be used for high availability.

- Type IV. Active data with frequent accesses and frequent changes

  Examples are Entity EJBs data, session data, and application data.

Data high availability is more difficult to implement than process high availability. Most importantly, data high availability and process high availability are both needed to complete the task. Data high availability is essential in most applications. It does not make any sense for a business to run a process if that process cannot access required data. For example, there is little value in running a stock brokerage system if stock data and trading session data are unavailable. It does not help to have the WebSphere EJB container process available if Entity EJBs cannot access their data states, or to have the WebSphere Web container process if servlets cannot access needed HTTP session states.

We have discussed aspects and techniques for building an end-to-end highly available WebSphere production system. Through these techniques, we can achieve both data high availability and process high availability for the WebSphere system.

Part 7

Appendixes

A

Handling StaleConnectionException

This appendix provides details and code samples for handling the StaleConnectionException thrown by WebSphere. Generally, when a StaleConnectionException is caught, the transaction in which the connection was involved needs to be rolled back and a new transaction begun with a new connection. Details on how to do this can be broken down into three categories:

- A connection in auto-commit mode.
- A connection not in auto-commit and transaction begun in the same method as database access.
- A connection not in auto-commit and transaction begun in a different method from database access.

Connections in auto-commit mode

By default, any connection obtained from a one-phase datasource (implementing javax.sql.ConnectionPoolDataSource) is in auto-commit mode, when there is no scoping transaction. When in auto-commit mode, each database action (statement) is executed and committed in a single database transaction. Servlets often use connections in auto-commit, because transaction semantics are not necessary.

Enterprise applications do not usually use auto-commit connections, because they frequently require multiple statements to be executed, and serially committing each would be quite cumbersome. Auto-commit can be explicitly disabled by calling setAutoCommit() on a Connection object. When a StaleConnectionException is caught from a connection in auto-commit mode, recovery is a simple matter of closing all of the associated JDBC resources and retrying the operation with a new connection.

However, in some cases the cause of a database outage might be transient. In these cases, adding a delay in the retry logic can allow a database service to be restored. The number of retries as well as any delay should be small, so as to keep a client from waiting an inordinate amount of time. Sample code is shown in Example A-1.

Example: A-1 Sample code for connections in auto-commit mode

```
public void myConnPool() throws java.rmi.RemoteException {
   // retry indicates whether to retry or not
   // numOfRetries states how many retries have been attempted
   boolean retry = false;
   int numOfRetries = 0;
   java.sql.Connection conn = null;
   java.sql.Statement stmt = null;
   do {
      try {
         //Assumes that a datasource has already been obtained from JNDI
         conn = ds.getConnection();
         stmt = conn.createStatement();
         stmt.execute(
            "INSERT INTO ORG VALUES (10, 'Pacific', '270', 'Western',
'Seattle')");
         retry = false;
      } catch (com.ibm.websphere.ce.cm.StaleConnectionException sce) {
         //if a StaleConnectionException is caught rollback and retry the
action
         if (numOfRetries < 2) {
            retry = true;
            numOfRetries++;
            // add an optional pause
```

```
                Thread.sleep(10000);
            } else {
                retry = false;
            }
        } catch (java.sql.SQLException sqle) {
            //deal with other database exception
        } finally {
            //always cleanup JDBC resources
            try {
                if (stmt != null)
                    stmt.close();
            } catch (java.sql.SQLException sqle) {
                //usually can ignore
            }
            try {
                if (conn != null)
                    conn.close();
            } catch (java.sql.SQLException sqle) {
                //usually can ignore
            }
        }
    } while (retry);
}
```

Connections with auto-commit disabled

When a connection has auto-commit disabled, multiple database statements can be executed in the same transaction. Because transactions tend to use a significant number of resources, fewer transactions result in better performance. Therefore, if an application requires executing multiple statements, best practice is to disable auto-commit, and use transactions to group a number of statements into one unit of work. Keep in mind that if a transaction has a very large number of statements, the database can experience memory resource issues.

Transactions started in the same method

If a transaction is begun in the same method as the database access, recovery is straightforward and similar to the case of using a connection in auto-commit mode. When a StaleConnectionException is caught, the transaction is rolled back and the method retried. As was the case with connections where

auto-commit is enabled, the time delay between, as well as the number of, retries should be limited. This is illustrated in Example A-2.

Example: A-2 Sample code for transactions started in same method as database access

```
do {
   try {
      //begin a transaction
      tran.begin();
      //Assumes that a datasource has already been obtained from JNDI
      conn = ds.getConnection();
      conn.setAutoCommit(false);
      stmt = conn.createStatement();
      stmt stmt.execute(
         "INSERT INTO ORG VALUES (10, 'Pacific', '270', 'Western',
'Seattle')");
      tran.commit();
      retry = false;
   } catch (com.ibm.websphere.ce.cm.StaleConnectionException sce) {
      //if a StaleConnectionException is caught rollback and retry the action
      try {
         tran.rollback();
      } catch (java.lang.Exception e) {
         //deal with exception in most cases, this can be ignored
      }
      // deal with other database exceptions and clean up as before
   }
}
```

Transactions started in a different method from database access

When a transaction is begun in a method different from where a database connection is accessed, an exception needs to be thrown from the database access method, to indicate a failure. In an ideal situation, a method can throw an application-defined exception, indicating that the transaction logic can be retried. However this is not always allowed, and often a method is defined only to throw a particular exception. This is the case, for example, in the ejbLoad and ejbStore methods on an enterprise bean.

Reference

The WebSphere V6 InfoCenter contains verbose information about this topic. Search for *StaleConnectionException* to find examples and additional information.

B

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/SG246688

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select **Additional materials** and open the directory that corresponds with the redbook form number, SG24-6688.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

File name	*Description*
TM_NoOP.zip	Zipped code samples for Transaction Manager NoOP scenario
ME_NoOP.zip	Zipped code samples for messaging engine NoOP scenario
Trade_iSeries.zip	Zipped code, installation instructions, and installation scripts for Trade 6 in an iSeries WebSphere environment

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space:	25 MB minimum (for Trade_iSeries.zip), 1 MB for other zipped files
Operating System:	Windows, Linux, UNIX, OS/400

How to use the Web material

TM_NoOP.zip and ME_NoOP.zip

- Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.
- Use the scripts as described in the chapters from Part 5, “Using external clustering software” on page 283.
- The HAMonitorWeb application (= the hamonitor.ear file) should be installed on the cluster using all default settings.

Trade_iSeries.zip

- Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder. This gives you 2 files:
  - trade51dbz
  - tradeinstall.zip
- trade51dbz: This is an OS/400 save file that contains the database needed for Trade 6. First ftp this file to the iSeries environment into an existing save file, then restore it.
- tradeinstall.zip: Extract the files into the /home IFS directory on the system where you wish to install Trade 6. This creates the /home/tradeinstall directory.

- In QSHELL, cd to /home/tradeinstall. From here, run the command **installTrade60.sh**.

  The install script asks a few questions regarding your environment, answer them as follows:

  - Select the backend database type: iSeriesNative
  - Enter the database schema: trade51dbz
  - Enter the database user name: Enter a valid user profile
  - Enter the database password: The password for the user profile

  The script then stops and restarts the server, and you are ready to go.

- Next, fine-tune the application server with the following command:

  ```
  tuneWAS60.jacl server trade60
  ```

- Finally, verify the maximum heap size of the application server JVM. It should be set to 0 (which is the default setting in WebSphere for iSeries and means no limit).

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 611. Note that some of the documents referenced here might be available in softcopy only.

- *IBM WebSphere V6 Scalability and Performance Handbook*, SG24-6392
- *IBM WebSphere V6 Planning and Design Handbook*, SG24-6446
- *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451
- *WebSphere Application Server V6: Security Handbook*, SG24-6316
- *Logical Partitions on the IBM PowerPC: A Guide to Working with LPAR on POWER5 for IBM @server i5 Servers*, SG24-8000
- *Clustering and IASPs for Higher Availability on the IBM @server iSeries Server*, SG24-5194
- *Advanced POWER Virtualization on IBM @server p5 Servers: Introduction and Basic Configuration*, SG24-7940
- *WebSphere v6 HA for z/OS*, SG24-6850
- *WebSphere MQ in a z/OS Parallel Sysplex Environment*, SG24-6864
- *Securing NFS in AIX An Introduction to NFS v4 in AIX 5L Version 5.3*, SG24-7204
- *IBM TotalStorage Enterprise Storage Server Implementing ESS Copy Services in Open Environments*, SG24-5757
- *A Practical Guide to DB2 UDB Data Replication V8*, SG24-6828
- *Understanding LDAP Design and Implementation*, SG24-4986

Online resources

These Web sites and URLs are also relevant as further information sources:

- WebSphere V6 InfoCenter:

  http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp

- IBM WebSphere Application Server Network Deployment V6 hardware and software requirements:

  http://www.ibm.com/software/webservers/appserv/was/requirements/

- *WebSphere Application Server support* Web site:

  http://www.ibm.com/software/webservers/appserv/was/support/

- WebSphere Application Server for z/OS support Web site:

  http://www.ibm.com/software/webservers/appserv/zos_os390/support

- *IBM WebSphere Application Server - Clarification of configurations support*:

  http://www.ibm.com/support/docview.wss?rs=180&context=SSEQTP&uid=swg27004311

- *IBM WebSphere Application Server Network Deployment V6: Installing your application serving environment*:

  ftp://ftp.software.ibm.com/software/webserver/appserv/library/v60/wasv60nd_gs.pdf

- White paper *Transactional high availability and deployment considerations in WebSphere Application Server V6*:

  http://www.ibm.com/developerworks/websphere/techjournal/0504_beaven/0504_beaven.html

- Trade 6 download:

  http://www.ibm.com/software/webservers/appserv/was/performance.html

- Sample **wsadmin** scripts for WebSphere administration tasks:

  http://www.ibm.com/developerworks/websphere/library/samples/SampleScripts.html

- Technote *Stop all WebSphere Application Server-related Java processes before using the Update Installer for WebSphere software*:

  http://www.ibm.com/support/docview.wss?rs=180&uid=swg21199141

- File System Locking Protocol Test:

  http://www.ibm.com/support/docview.wss?rs=180&uid=swg24010222

- *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

- White paper *Server Clusters for High Availability in WebSphere Application Server Network Deployment Edition 5.0*:

  http://www.ibm.com/support/docview.wss?uid=swg27002473

- Capacity on Demand Web site:

  http://www.ibm.com/servers/eserver/about/cod/

- *Parallel Sysplex Cluster Technology:*

  http://www.ibm.com/servers/eserver/zseries/pso/sysover.html

- *Leveraging z/OS TCP/IP Dynamic VIPAs and Sysplex Distributor for higher availability:*

  http://www.ibm.com/servers/eserver/zseries/library/techpapers/gm130165.html

- iSeries Information Center:

  http://publib.boulder.ibm.com/html/as400/infocenter.html

- iSeries cross-site mirroring fixes:

  http://www-912.ibm.com/s_dir/slkbase.nsf/ibmscdirect/

- *iSeries and High Availability An e-Business Perspective*:

  http://www.ibm.com/servers/eserver/iseries/software/websphere/wsappserver/product/iSeriesAndHa.pdf

- IBM TotalStorage Web site:

  http://www.storage.ibm.com

- DB2 Information Center:

  http://publib.boulder.ibm.com/infocenter/db2help/index.jsp

- DB2 Information Management Software Information Center for z/OS Solutions:

  http://publib.boulder.ibm.com/infocenter/dzichelp/index.jsp

- *An Overview of High Availability and Disaster Recovery for DB2 UDB* by Monty Wright:

  http://www.ibm.com/developerworks/db2/library/techarticle/0304wright/0304wright.html

- DB2 HADR:

  http://publib.boulder.ibm.com/infocenter/db2help/index.jsp?topic=/com.ibm.db2.udb.doc/admin/c0011267.htm

- *Understanding high availability with WebSphere MQ* by Mark Hiscock and Simon Gormley:

  http://www3.software.ibm.com/ibmdl/pub/software/dw/wes/pdf/0505_hiscock_MQ_HA.pdf

- HACMP for AIX 5L Web site:

  http://www.ibm.com/servers/aix/products/ibmsw/high_avail_network/hacmp.html

- HACMP documentation:

  http://www.ibm.com/servers/eserver/pseries/library/hacmp_docs.html

- *MQSeries for AIX - Implementing with HACMP Version 2.0*:

  ftp://ftp.software.ibm.com/software/integration/support/supportpacs/individual/mc63.pdf

- *IBM DB2 Universal Database Enterprise Edition for AIX and HACMP/ES*:

  ftp://ftp.software.ibm.com/software/data/pubs/papers/db2ee-aixhacmp.pdf

- IBM Tivoli System Automation for Multiplatforms Web site:

  http://www.ibm.com/software/tivoli/products/sys-auto-linux/

- *IBM Tivoli System Automation for Multiplatforms Guide and Reference*:

  http://publib.boulder.ibm.com/tividd/td/ITSAFL/SC33-8210-03/en_US/PDF/halgre11.pdf

- IBM Tivoli System Automation for Multiplatforms - Downloads:

  http://www.ibm.com/software/tivoli/products/sys-auto-linux/downloads.html

- *Quality busters: Forget the environment*:

  http://www.ibm.com/developerworks/web/library/wa-qualbust1/index.html

- Veritas Web site:

  http://www.veritas.com

- *VERITAS Cluster Server Agent 3.5 for WebSphere MQ Installation and Configuration Guide*:

  http://ftp.support.veritas.com/pub/support/products/ClusterServer_UNIX/273084.pdf

- *VERITAS Cluster Server Enterprise Agent 4.0 for DB2 Installation and Configuration Guide*:

  http://ftp.support.veritas.com/pub/support/products/ClusterServer_UNIX/270374.pdf

- Sun Cluster software:

  http://www.sun.com/clusters

- Sun Cluster Software Installation and Sun Cluster Software Administration guides:

  http://docs.sun.com/app/docs

- *Sun Cluster Data Service for WebSphere MQ Integrator Guide for Solaris OS*:

  http://docs.sun.com/app/docs/doc/817-4580

- *IBM DB2 Universal Database and High Availability on Sun Cluster 3.x*:

  ftp://ftp.software.ibm.com/software/data/pubs/papers/suncluster.pdf

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

Index

C

D

E

F

G

H

I

N

O

P

Q

R

S

T

U

V

W

X

Z

WebSphere Application Server Network Deployment V6: High Availability Solutions

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

WebSphere Application Server Network Deployment V6: High Availability Solutions

WebSphere Handbook Series

Explore WebSphere HA options

Learn about external clustering solutions

This IBM Redbook discusses the high availability aspects of IBM WebSphere Application Server Network Deployment V6 and high availability of related components, such as the Web servers or directory servers. This book discusses in detail:

- High availability concepts.
- WebSphere Application Server clustering considerations, the failover process, the WebSphere HAManager, and WebSphere component's reactions to failures.
- High availability system administration, such as application management, hardware replacement or upgrade, and software upgrades.
- WebSphere Node Agent and Deployment Manager high availability using external clustering software solutions such as IBM HACMP, IBM Tivoli System Automation (TSA), VERITAS Cluster Server, and Sun Cluster.
- High availability considerations and differences when using WebSphere in iSeries environments and zSeries environments.
- End-to-end WebSphere system high availability involving WebSphere MQ, Web servers, Load Balancer, firewalls, and LDAP servers.

The book also gives an introduction into how to backup and recover a Network Deployment configuration.

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

SG24-6688-00 ISBN 0738492507