

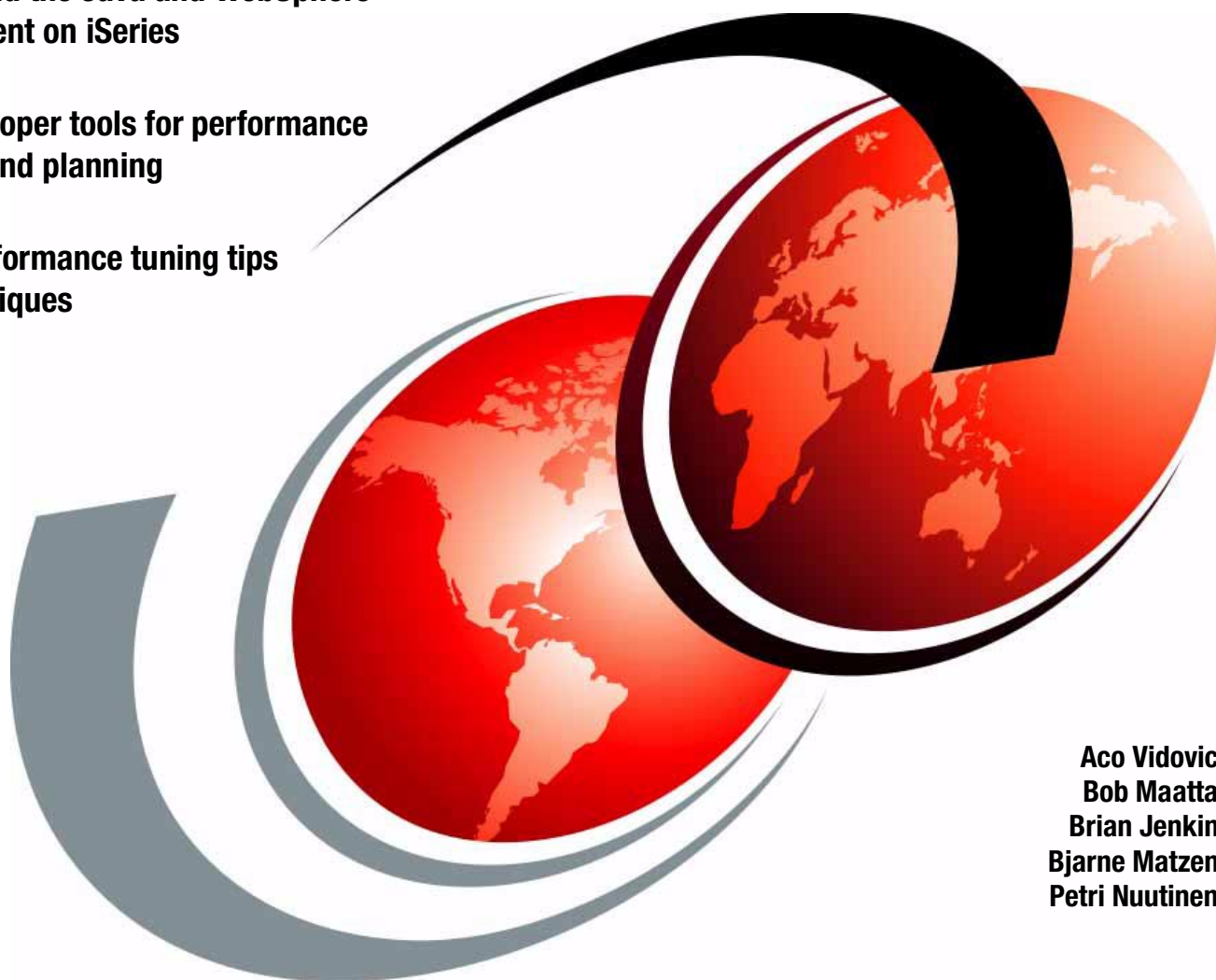
# Java and WebSphere Performance

on IBM @server iSeries Servers

Understand the Java and WebSphere  
environment on iSeries

Use the proper tools for performance  
analysis and planning

Learn performance tuning tips  
and techniques



Aco Vidovic  
Bob Maatta  
Brian Jenkin  
Bjarne Matzen  
Petri Nuutinen

**Redbooks**





International Technical Support Organization

**Java and WebSphere Performance  
on IBM @server iSeries Servers**

February 2002

**Take Note!** Before using this information and the product it supports, be sure to read the general information in “Special notices” on page vii.

### **First Edition (February 2002)**

This edition applies to Version 3, Release 5 and Version 4, Release 0 of WebSphere Application Server Advanced Edition, program numbers 5733-WA3 and 5733-WA4, for use with the Version 5, Release 1 of OS/400.

Comments may be addressed to:  
IBM Corporation, International Technical Support Organization  
Dept. JLU Building 107-2  
3605 Highway 52N  
Rochester, Minnesota 55901-7829

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2002. All rights reserved.

Note to U.S Government Users - Documentation related to restricted rights - Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

# Contents

<b>Contents</b> .....	iii
<b>Special notices</b> .....	vii
<b>IBM trademarks</b> .....	viii
<b>Preface</b> .....	ix
The team that wrote this redbook .....	ix
Special notice .....	xi
Comments welcome .....	xi
<b>Chapter 1. Introduction</b> .....	1
1.1 Client performance .....	2
1.2 Network performance .....	3
1.3 Server performance .....	3
<b>Chapter 2. The Java execution environment on iSeries servers</b> .....	7
2.1 Java components on iSeries .....	8
2.1.1 Java implementation components .....	8
2.1.2 Java runtime components .....	9
2.2 Java Transformer and the just-in-time compiler (JIT) .....	9
2.2.1 Creating a Java program .....	10
2.2.2 Difference between direct execution and JIT .....	11
2.3 Java garbage collection .....	11
2.3.1 Understanding garbage collection .....	12
2.3.2 Problems with garbage collection .....	13
2.4 IBM Toolbox for Java .....	13
<b>Chapter 3. Performance methodology and tools</b> .....	15
3.1 Setting performance objectives .....	16
3.2 General tuning process and performance strategy .....	17
3.3 Performance problem determination process .....	18
3.4 Overview of iSeries performance measurement tools .....	19
3.4.1 Collecting performance data .....	19
3.4.2 Creating performance data .....	22
3.5 System level tools .....	25
3.5.1 Management Central .....	25
3.5.2 WRKSYSSTS command .....	26
3.5.3 WRKDSKSTS command .....	27
3.5.4 WRKACTJOB command .....	30
3.5.5 WRKSYSACT command .....	34
3.5.6 Performance Tools for iSeries licensed program .....	35
3.5.7 Using the system performance tools .....	37
3.6 WebSphere performance tools .....	38
3.6.1 WebSphere Application Server Resource Analyzer .....	38
3.7 Java and application-level tools .....	39
3.7.1 DMPJVM command .....	40
3.7.2 Performance Explorer (PEX) .....	41
3.7.3 Performance Trace Data Visualizer for iSeries (PTDV) .....	42

3.7.4 Database Monitor for iSeries. . . . .	51
3.7.5 SQL Visual Explain . . . . .	53
<b>Chapter 4. Tuning iSeries for a WebSphere or Java environment . . . . .</b>	<b>57</b>
4.1 iSeries performance behavior . . . . .	58
4.2 Verifying the state of Licensed Program Products . . . . .	58
4.3 OS/400 system tuning . . . . .	60
4.3.1 Manual tuning versus automatic tuning. . . . .	60
4.3.2 Verifying the settings of system values. . . . .	61
4.3.3 Creating a separate memory pool for a subsystem . . . . .	67
4.3.4 Setting the memory pool sizes and activity levels manually . . . . .	70
4.3.5 Common WebSphere Application Server and Java jobs . . . . .	73
4.3.6 Working with the prestart jobs. . . . .	74
4.4 Java system environment . . . . .	81
4.4.1 Java thread priority . . . . .	81
4.4.2 Time slice setting . . . . .	81
4.5 AnyNet . . . . .	81
<b>Chapter 5. Tuning HTTP server and WebSphere Application Server. . . . .</b>	<b>83</b>
5.1 Tuning the HTTP server . . . . .	84
5.1.1 Caching. . . . .	84
5.1.2 Persistent connections . . . . .	86
5.1.3 Secure Sockets Layer (SSL). . . . .	87
5.1.4 Maximum number of active HTTP server threads. . . . .	88
5.1.5 IBM HTTP Server (powered by Apache). . . . .	91
5.2 Direct execution (DE) versus JIT mode . . . . .	91
5.3 WebSphere queueing network . . . . .	93
5.3.1 Closed queues versus open queues. . . . .	93
5.3.2 Queue settings . . . . .	94
5.3.3 Tuning the queues . . . . .	94
5.4 Garbage collection initial size . . . . .	97
5.5 Class auto reload and reload interval . . . . .	98
5.6 Ping timeout and ping initial timeout . . . . .	99
5.7 Standard output and standard error . . . . .	100
5.8 Session Manager . . . . .	101
5.9 Security . . . . .	102
5.10 EJB container . . . . .	102
5.11 Prepared statement cache . . . . .	103
5.12 WebSphere plug-in to HTTP server . . . . .	104
5.13 JSP processing engine . . . . .	104
5.14 Tuning parameters to leave untouched. . . . .	105
<b>Chapter 6. Java and WebSphere application design . . . . .</b>	<b>111</b>
6.1 Java design . . . . .	112
6.2 Instruction execution: Coding considerations . . . . .	112
6.2.1 Instantiation results in garbage collection. . . . .	112
6.2.2 String manipulation . . . . .	113
6.2.3 Method resolution . . . . .	113
6.2.4 Exception handling . . . . .	113
6.2.5 Java Native Interface (JNI) . . . . .	113
6.2.6 Thread creation. . . . .	114
6.2.7 Variable scope . . . . .	114
6.2.8 Remote AWT . . . . .	114
6.3 Accessing system services: Database operations . . . . .	114

6.3.1 IBM Toolbox for Java driver versus iSeries Developer Kit for Java driver . . . . .	116
6.3.2 JDBC through a native driver . . . . .	117
6.4 Coding considerations with JDBC . . . . .	118
6.4.1 Data conversion . . . . .	118
6.4.2 Retrieve only the necessary columns . . . . .	119
6.4.3 Use get...(columnIndex) instead of get...(columnName). . . . .	119
6.4.4 Avoid using AutoCommit. . . . .	119
6.4.5 Consider using batch insert . . . . .	119
6.4.6 Use prepared statements . . . . .	120
6.5 Coding considerations with DDM (record level access) . . . . .	120
6.5.1 Minimizing open and close . . . . .	120
6.5.2 Blocking on READ_ONLY and WRITE_ONLY . . . . .	120
6.5.3 Downloading a small file using the readAll() method . . . . .	121
6.6 Distributed Program Call (DPC) . . . . .	121
6.7 Coding considerations for servlets and JSPs . . . . .	121
6.7.1 Do not store large objects in HttpSession . . . . .	121
6.7.2 Release HttpSession when finished . . . . .	122
6.7.3 Do not create HttpSession in JSPs by default. . . . .	122
6.7.4 Do not use SingleThreadModel . . . . .	122
6.7.5 Use the HttpServlet Init() method . . . . .	122
6.7.6 Minimize or eliminate use of System.out.println() . . . . .	122
6.7.7 Don't use Beans.instantiate() to create new bean instances . . . . .	123
6.7.8 Use and reuse DataSources for JDBC connections . . . . .	123
6.7.9 Release JDBC resources when done . . . . .	124
6.8 Coding considerations for EJBs . . . . .	124
6.8.1 Access entity beans from session beans . . . . .	125
6.8.2 Reuse EJB homes . . . . .	125
6.8.3 Remove stateful session beans when finished . . . . .	125
6.9 EJB deployment considerations . . . . .	125
6.9.1 Use read-only methods where appropriate. . . . .	125
6.9.2 Reduce the transaction isolation level where appropriate. . . . .	126
<b>Chapter 7. Case study . . . . .</b>	<b>127</b>
7.1 System used in this case study. . . . .	128
7.2 Application used in this case study . . . . .	128
7.2.1 Overview of the OrderEntry application . . . . .	128
7.2.2 Application architecture. . . . .	129
7.2.3 A customer transaction . . . . .	133
7.2.4 Client application design . . . . .	134
7.2.5 OrderEntry application database layout . . . . .	139
7.3 Tuning the OS/400 . . . . .	142
7.3.1 Examining and tuning the QSYSWRK subsystem . . . . .	143
7.3.2 Examining and tuning the QEJBSBS subsystem . . . . .	144
7.3.3 Summary of system tuning . . . . .	145
7.4 Tuning WebSphere Application Server. . . . .	145
7.4.1 Direct Execution versus JIT . . . . .	147
7.4.2 Tuning WebSphere Application Server queues . . . . .	147
7.4.3 Garbage collection initial size . . . . .	149
7.4.4 Other tuning techniques . . . . .	150
7.5 Tuning the application . . . . .	150
7.5.1 Analyzing the application . . . . .	151
7.5.2 Further improving the application . . . . .	155
7.6 Changing to WebSphere 4.0 . . . . .	164

7.7 Summary .....	164
<b>Chapter 8. Scaling the WebSphere environment</b> .....	167
8.1 Scalability objectives .....	168
8.2 Scaling a solution .....	169
8.2.1 Workload patterns .....	169
8.2.2 Scalability in practice .....	171
8.2.3 Six steps to scaling your solution .....	172
8.3 Specific iSeries scaling considerations .....	175
8.3.1 Overview .....	176
8.3.2 Integrated DB2 database .....	176
8.3.3 iSeries WebSphere scalability overview and key concepts .....	176
8.3.4 Specific iSeries WebSphere scaling .....	177
8.3.5 Current iSeries JDBC driver limitations .....	180
<b>Chapter 9. Sizing and capacity planning</b> .....	181
9.1 Why do performance and capacity planning .....	182
9.2 Generic performance process and methodology .....	183
9.2.1 Process steps .....	184
9.3 Capacity planning and workload estimation tools .....	186
9.3.1 BEST/1 .....	187
9.3.2 IBM Workload Estimator .....	205
9.3.3 PM/400 .....	210
9.4 Disk arm sizing .....	210
9.5 Stress testing the WebSphere application .....	210
9.5.1 General process for load and stress testing .....	210
9.5.2 Load generation tools .....	211
9.6 Sizing an application under development .....	211
9.6.1 Validating the capacity estimates .....	211
<b>Related publications</b> .....	213
IBM Redbooks .....	213
Other resources .....	213
Referenced Web sites .....	213
How to get IBM Redbooks .....	215
IBM Redbooks collections .....	215
<b>Abbreviations and acronyms</b> .....	217
<b>Index</b> .....	219



# Special notices

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter 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 the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.


Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

# IBM trademarks

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

e (logo)® 

AnyNet®

APPN®

AS/400®

AS/400e™

Balance®

BookMaster®

COBOL/400®

DB2®

DB2 Universal Database™

DRDA®

e (logo)®

Redbooks Logo 

IBM®

iSeries™

Net.Commerce™

OS/400®

Perform™

Redbooks™

S/390®

SecureWay®

SP™

VisualAge®

WebSphere®

## Other company trademarks

The following terms are trademarks of other companies:

Tivoli, Manage. Anything. Anywhere., The Power To Manage., Anything. Anywhere., TME, NetView, Cross-Site, Tivoli Ready, Tivoli Certified, Planet Tivoli, and Tivoli Enterprise are trademarks or registered trademarks of Tivoli Systems Inc., an IBM company, in the United States, other countries, or both. In Denmark, Tivoli is a trademark licensed from Kjøbenhavns Sommer - Tivoli A/S.

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

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

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

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

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

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

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

# Preface

This IBM Redbook provides tips, techniques, and methodologies for working with Java and WebSphere Application Server performance-related issues with a specific focus on iSeries servers. The intended audience includes iSeries system performance experts and WebSphere Application Server and Java developers. This book attempts to provide you with a comprehensive and accessible resource that has gathered information usually found in a number of disparate sources and placed them in a single reference.

The performance measurements that are published in this redbook are done by using a Java application that runs on WebSphere Application Server Release 3.5.4 and 4.0 and OS/400 V5R1. This redbook includes these topics:

- ▶ An introduction to the Java and WebSphere Application Server execution environment on the iSeries
- ▶ OS/400 work management and tuning in Java/WebSphere Application Server environment
- ▶ Tuning WebSphere Application Server and Web server (HTTP server)
- ▶ Performance tips for application development and deployment
- ▶ Sizing and capacity planning
- ▶ Recommended tools to use for performance analysis and planning
- ▶ A case study that shows an example of performance management methodology and tuning

**Note to reader:** Throughout this redbook, to emphasize what you should look for in the windows and displays, we use reverse imaging on the text, as shown here **1234**.

## 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 Rochester Center.



**Aco Vidovic** is a Senior IT Specialist in IBM ITSO Rochester Center on assignment from IBM Slovenia. Before joining the ITSO in 1999, he worked in the AS/400 area for 11 years in a variety of roles from second level technical support to marketing. In the ITSO, Aco teaches extensively and writes IBM Redbooks about WebSphere and Java performance and implementation of Enterprise Resource Planning applications on the iSeries server.



**Bob Maatta** is Consulting Software Engineer at the IBM International Technical Support Organization, Rochester Center. He is the ITSO technical leader for iSeries e-business application development. He writes extensively and develops and teaches IBM classes worldwide on all areas of iSeries client/server and e-business application development. He has worked on numerous computer platforms including S/390, S/38, AS/400, and personal computers. In his current assignment, he specializes in Java programming and the IBM WebSphere Application Server. He is a Sun Certified Java Programmer and a Sun Certified Java Developer.



**Bjarne Matzen** is Director of Product Development for Mid-Comp International, an Australian-based IBM Business Partner. He has more than 16 years experience with application development and systems integration, spanning from industrial robotics to commercial logistics systems (ERP). His area of expertise includes the iSeries server with RPG, C, and Java, as well as Object Oriented Programming on various other platforms. He was the original author of the Snapshot/400 performance monitoring system, which is widely used by IBM iSeries customers worldwide. His current responsibilities include the development of a Java-based ERP.



**Brian Jenkin** is a Senior IT Specialist in New Zealand. He has more than 20 years of experience in application development, performance, and capacity planning within multiple environments. He holds degrees in Electrical Engineering and Pure Mathematics from Canterbury University, New Zealand. He is currently responsible for the performance of the Integrated Customer Management System (ICMS), which is IBM's multi-service Customer Care & Billing (CC&B) solution for telecommunications companies, supporting wireline, wireless, cable TV, and data services.



**Petri Nuutinen** is a Systems Support Specialist in IBM Finland who started working with S/38 in 1982. In 1987, he joined IBM and was held responsible for the technical support of the AS/400 MRI translation process. His technical specialties are work management and performance tuning, and he has participated in several performance-related residencies and redbook projects since 1991. Currently he divides his working time between pre and post sales and billable services.

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

Jeremy Arnold  
 Jim Beck  
 Jim Cook  
 Debbie Hatt  
 Leonardo Llamas  
 Scott Moore  
 Rick Peterson  
 Curtis Schmelling  
 Clark Scholten  
 Brian Smith

Lisa Wellman  
Neil Willis  
Mike Zanoni  
**IBM Rochester**

Richard Nesbitt  
**IBM Raleigh**

Monica Echeverri  
**IBM Colombia**

Gaia Banchelli  
**IBM Italy**

We also acknowledge the efforts and work of many others, too numerous to mention, whose IBM Redbooks, white papers, articles, and presentations have provided us with background ideas, and in some cases, materials with which to create this document.

## Special notice

This publication is intended to help Java developers and performance analysts to achieve the best possible performance results from Java code running on the iSeries server. The information in this publication is not intended as the specification of any programming interfaces that are provided by Java, WebSphere, or OS/400. See the PUBLICATIONS section of the IBM Programming Announcement for WebSphere and OS/400 for more information about what publications are considered to be product documentation.

## Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Please 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](http://ibm.com/redbooks)
- ▶ Send your comments in an Internet note to  
[redbook@us.ibm.com](mailto:redbook@us.ibm.com)
- ▶ Mail your comments to address on Page ii



# Introduction

Before you start tuning and managing the performance of any system, you must understand your system's components and how they can affect the overall performance. To tune performance in a WebSphere Application Server environment, you should know every part of your system that is involved in this process. Maintaining good performance requires you to understand, plan, manage performance policies, and keep track of them. There are three major components of Web environments, each with its own performance characteristics and requirements:

- ▶ Client
- ▶ Server
- ▶ Network

Figure 1-1 shows the performance components of a typical WebSphere Application Server-based application environment.

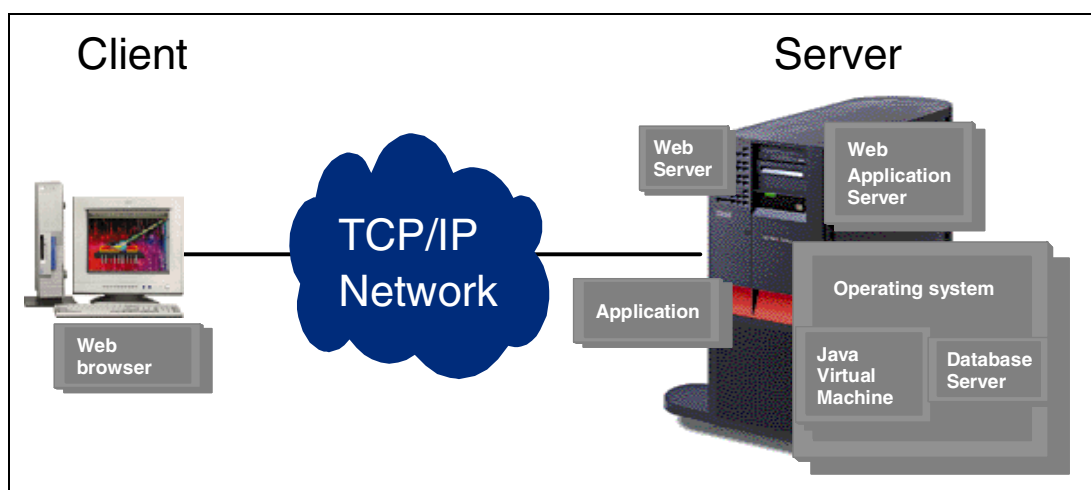


Figure 1-1 Performance components

The client in Figure 1-1 is using a Web browser (a user interface) and a TCP/IP network (Internet, intranet, or extranet) to access the application that runs on a server. Several other layers of middleware necessary to support the application environment run on the server:

- ▶ **Java Virtual Machine (JVM)** to provide runtime environment for Java programs. On iSeries servers, JVM is part of the operating system, OS/400.
- ▶ **Database server** with a relational database management system. On iSeries servers, the database server is DB2 Universal Database for iSeries, which is also a part of OS/400.
- ▶ **HTTP server** (also known as the Web server): The Hypertext Transfer Protocol (HTTP) server allows an iSeries server attached to a TCP/IP network to provide objects to any Web browser. After the Web browser and HTTP server establish a connection, the Web browser sends a request, which is received and processed by the HTTP server. If the request includes running an application, the HTTP server passes the request to WebSphere Application Server, which provides the environment for the Web application. The output of the application is sent back to the HTTP server, which formats the data and sends it to the Web browser, which displays it to the end user. After this, the connection between the browser and the HTTP server is terminated.
- ▶ **Web application server** provides the essential e-business functions of handling transactions and extending back-end business data and applications to the Web. There are several Web application servers on the market; the IBM Web application server is WebSphere Application Server, which we cover in this redbook. WebSphere Application Server supports servlets, JavaServer Pages, and Enterprise JavaBeans. It provides a Java-based servlet engine that's built on top of the iSeries native Java Virtual Machine. It implements the Java servlet API support, which is defined by Sun Microsystems. If you write to the Java servlet API standard, your application is portable across any operating system and any environment that supports servlets.
- ▶ **Application:** The set of programs that perform business logic, such as order entry, general ledger, and so on.

A problem in any of these areas may impact the overall performance. A problem may pertain to throughput, client processor speed, bandwidth, and resource utilization, to mention just a few factors.

## 1.1 Client performance

WebSphere Application Server-based applications tend to do most of the processing on the server side. Therefore, the role of a client (a workstation) is reduced, in most cases, to facilitate a user interface, typically a Web browser, such as Netscape Navigator or Microsoft Internet Explorer. From this point of view, the client's effect on the overall system performance is less significant than the performance of a network or a server. Still, the client contributes to the system's (good or bad) performance with the following client resources:

- ▶ **Hardware**
  - *Processor speed:* Slower clients, such as those with a 66 Mhz CPU speed, may experience performance degradation compared to faster clients with 400, 800, or 1000 Mhz clients. We recommend that you upgrade your client to use the fastest processor speeds.
  - *Memory:* Memory is an important factor since many Web-related tasks use large amounts of memory to be completed. For example, if you try to use cache on the clients, storing data in memory to retrieve it later is much faster than retrieving it from hard disk drives.
  - *Other hardware:* Every piece of hardware on your client is important. For example, access to data on your hard disk might not be fast enough, or your modem throughput might be too low. To maximize utilization of these resources, upgrade your hardware. In some cases (as with some modems), you can upgrade your hardware drivers.



- ▶ *Operating system*: Some operating systems have better performance in terms of Web browsing. Your operating system may be fast with communications, but may not be fast enough when loading Java applets. Some operating systems have enhanced capabilities to modify parameters, for example, to deal with the network. Some operating systems allow you to increase the frame size you want to use for your network. Others do not let you do this. To minimize response time, consider using a different operating system if you think this is the reason for the delay in the response time. Typically, however, the operating system should not cause a performance problem.
- ▶ *Web browser*: This is a main user interface for Web applications; therefore, consider optimizing your browser. The available Web browsers on the market have several different strengths and weaknesses. For example, you may want to take advantage of cache capabilities that most of the browsers have. We recommend that you cache as close to the client when possible. This helps you to reduce client requests to the server.

## 1.2 Network performance

Usually, the network has much more performance impact than the client. This is because of a wide variety of factors such as throughput, network traffic, and speed of communication links. These factors can be examined in more detail if you look at following network components:

- ▶ Routers
- ▶ LAN topology
- ▶ Link speed
- ▶ Modem throughput
- ▶ Packet filters
- ▶ Proxy
- ▶ Socks servers

The network is a dynamic environment; often you cannot control all its components. However, your company can act on its own network components and try to boost performance within company realms. For example, you may add more bandwidth or an additional leased line.

This redbook does not cover network performance in detail. For more information on network performance, refer to the following sources:

- ▶ *iSeries Performance Capabilities Reference Version 5, Release 1*:  
<http://www-1.ibm.com/servers/eserver/series/perfmgmt/resource.htm>
- ▶ *AS/400 HTTP Server Performance and Capacity Planning*, SG24-5645
- ▶ *AS/400 Communication Performance Investigation - V3R6/V3R7*, SG24-4895

## 1.3 Server performance

This redbook covers server performance. It includes such topics as performance measurement and its analysis, performance characteristics of WebSphere Application Server-based applications, capacity planning, and how to do sizing on WebSphere Application Server-based applications.

The server performance components are integrated in three major areas:

- ▶ Application
- ▶ Web server and Web application server
- ▶ System software and hardware

Figure 1-2 outlines the general composition of the server. However, you may find additional components in your own scenario. You need to fully understand and be aware of these components to attempt to acquire good server performance.

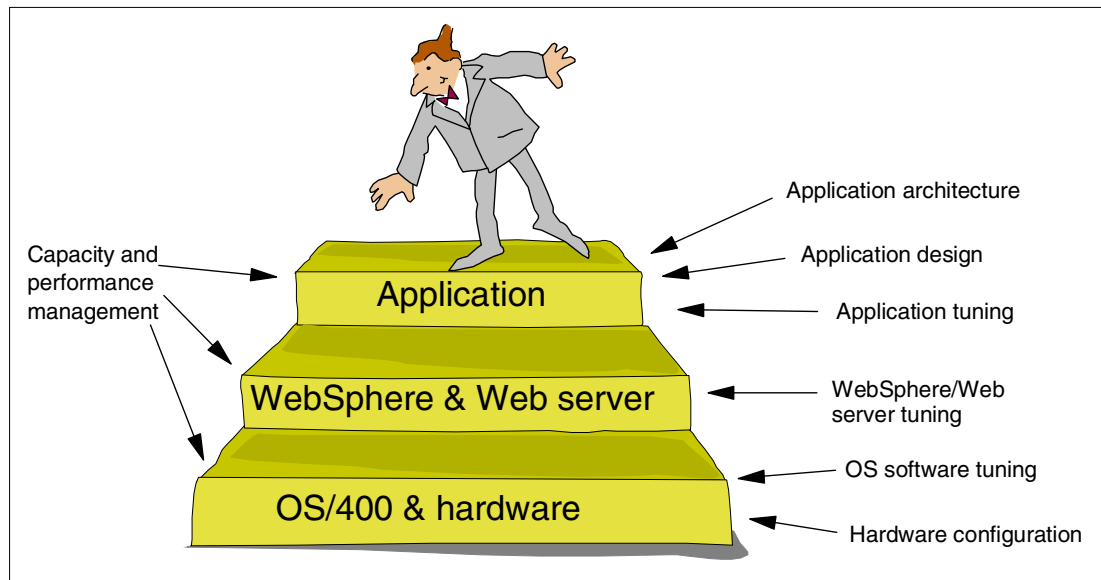


Figure 1-2 Performance as a balancing act

Performance is a balancing act, such that there is an optimum point at which the expenditure of time to measure matches the value of the improvements that may be obtained. Fundamentally, you are always looking for the most gains from the least effort.

The factors that influence the overall system and application performance include:

- ▶ **Hardware configuration:** Sufficient hardware resources must be present and available to the application.
- ▶ **System tuning options:** Ensure that the system and subsystems make the most efficient use of the hardware resources available.
- ▶ **Application**
  - *Application architecture:* Critical from the application perspective. The performance of a good architecture, even if implemented poorly, can frequently be overcome by application tuning, but the reverse does not apply.
  - *Application design:* Poor design or implementation can sometimes be overcome by tuning.

You should consider all these factors as a combined whole, when looking for performance opportunities.

## Server hardware resources

The following server hardware resources have the most significant impact on server performance:

- ▶ **CPU:** Since Java and WebSphere Application Server-based applications are typically CPU intensive, use the servers with fast CPUs. We recommend iSeries Models 270 and 8xx. Look for the servers with higher Mhz, not only higher CPW. If your iSeries server runs only Java and WebSphere Application Server workload, use a server with no interactive workload feature.

- ▶ **Main memory and memory cache:** Java and WebSphere Application Server applications require more memory (main storage) than traditional green-screen applications. Insufficient main memory can cause increased CPU and disk usage and lead to diminished throughput and response time. Make sure you have a server with a level 2 (L2) cache. The bigger the L2 cache is, the better.
- ▶ **Communication lines and communications IOPs:** iSeries servers use input/output processor (IOP) cards to minimize the amount of resource that the main CPU processors need to spend for peripheral operations, such as disk access and communications. IOP cards contain memory, processor, and input/output capabilities. Communications lines and IOPs may be overutilized during peak hours and, therefore, cause poor response time and throughput due to excessive queuing.
- ▶ **Disk arms, disk IOPs, and disk space:** As with any other system workload, it is important to have a sufficient number of disk arms (actuators) and a sufficient amount of free disk space. Once you have less than 50% of disk space free, disk service times start to increase. When you have less than 10% of your disk space free, the system starts sending you warning messages. For good performance, try to keep your disk arm utilization below 50% and disk IOP utilization below 60%.

## Java performance

Since applications based on the WebSphere Application Server are written in Java, this redbook spends a great deal of time on examining Java performance.

A lot has happened to the Java language since it was first released by Sun in May 1995. From originally being used as a language to create applets for interactive Web browser contents (and actually designed for embedding in devices such as toasters), Java has undergone a transformation into a full featured programming language for business applications. The promise of platform independence and an easy language syntax proved a great attraction to software developers.

Java's object oriented structure and ease of use also helped to make it the language of choice for many programmers working with e-business solutions. With this came an increased focus on component-based development. Also application servers became an important part of the infrastructure, providing standardized persistence mechanisms (JDBC and EJB) and user interaction (servlets and JSPs).

But with any metamorphosis comes a new set of problems. One of these for Java has always been performance. How do you address those Java-specific performance problems?

As hardware becomes faster and the Java technology matures, some of these issues become less important. However, solving a performance problem by simply adding more hardware (also known as the "brute force" method) is often not the best solution and can, in some cases, prove inefficient or simply not work. A good deployment methodology, system tuning, and application design should always be the first step to solving a performance problem.

Performance problems can at times be difficult to identify, and sometimes tweaking one setting can influence several others. In most cases, there is no simple answer, but careful measurements, good stress testing, and attention to coding rules can take some of the guesswork out of the performance of your Java application.

If you are developing an application, learn to include performance considerations early. Stress test and check the scalability of your early prototype. Keep stress testing on a regular basis as your application develops. This is one of the key methods for successful deployment with minimum surprises.

The next few chapters take you through some of the secrets to unlocking superior Java performance from your iSeries server.



## The Java execution environment on iSeries servers

The Java Virtual Machine (JVM) on the iSeries has a number of features that differentiate it from most other platforms. One of the most renowned is the asynchronous garbage collector, which provides superior and consistent performance. But there are other features that impact the overall performance of a Java application on the iSeries server.

This chapter introduces you to how Java applications are executed in the iSeries environment. It covers the following topics:

- ▶ Java execution environment
- ▶ Java Transformer and just-in-time compiler (JIT)
- ▶ Garbage collection
- ▶ IBM Toolbox for Java

## 2.1 Java components on iSeries

This section covers the basic components of the Java environment on the iSeries server. Understanding these components is essential to understanding how to tune the environment for performance.

### 2.1.1 Java implementation components

On the iSeries server, the JVM is implemented within SLIC, below the Technology Independent Machine Interface (TIMI) layer. It is an integral part of OS/400. When you install OS/400 V4R2 or a more recent release on your machine, you have installed a standard JVM on your system. In addition to OS/400, you can also install a number of Java Development Kits (JDKs) on your system. The base product for Java Development is JV1 (for example, licensed product 5722-JV1), which provides the Java compilers and run-time commands. You must also install at least one JV1 option to match your required JDK. Some examples are:

- ▶ JV1 option 4: JDK 1.1.8 for WebSphere Application Server Version 3.0
- ▶ JV1 option 3: JDK 1.2.2 for WebSphere Application Server Version 3.5
- ▶ JV1 option 5: JDK 1.3 for WebSphere Application Server Version 5

The SLIC implementation of the JVM uses the native thread support that came with the V4R2 release of OS/400. It also supports JNI calls to user-written ILE C or C++ native methods that are packaged in service programs (\*SRVPGM). Figure 2-1 provides an overview of the components in the Java implementation on the iSeries server.

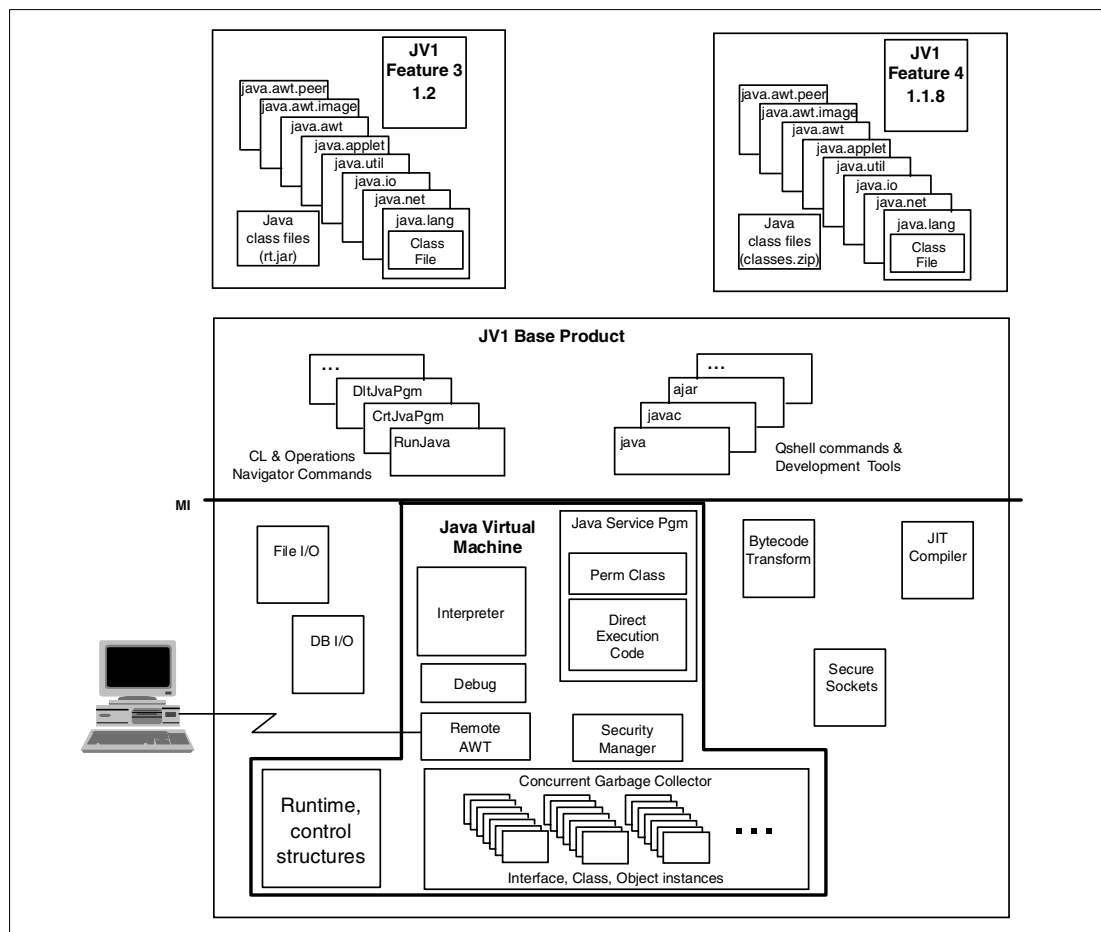


Figure 2-1 Java implementation under OS/400

## 2.1.2 Java runtime components

The Java Runtime Environment (JRE) starts whenever you enter the Run Java (RUNJVA) command or **java** command on the OS/400 command line. Because the Java environment is multithreaded, it is necessary to run the Java Virtual Machine in a job that supports threads, such as a batch immediate (BCI) job. Once the Java Virtual Machine starts, additional threads may start in which the garbage collector runs. Figure 2-2 illustrates the environment.

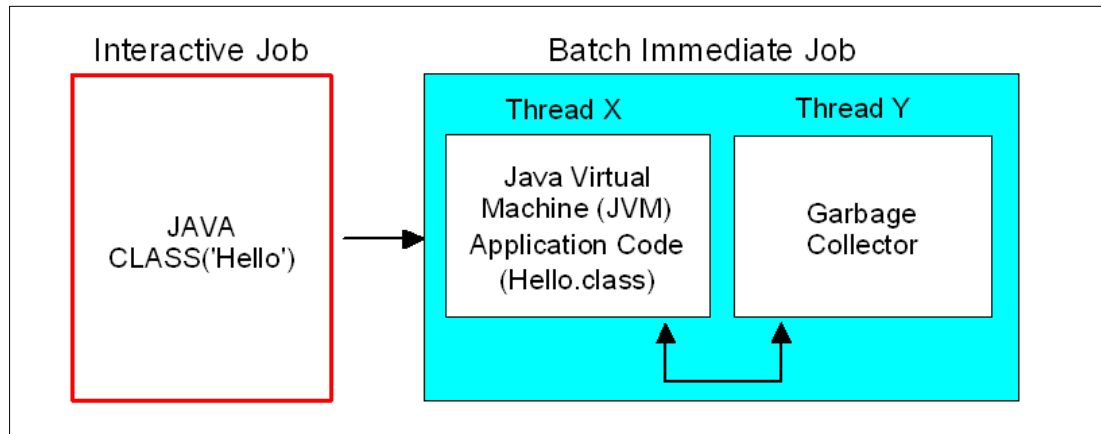


Figure 2-2 Java Runtime Environment

It is also possible to start the Java Runtime Environment by using the **java** command from within the Qshell Interpreter. In this environment, the Qshell Interpreter runs in a BCI job that is associated with an interactive job. The Java Runtime Environment starts in the job that is running the Qshell Interpreter.

## 2.2 Java Transformer and the just-in-time compiler (JIT)

Java classes are compiled into platform-independent bytecode. You can simply copy your Java class from another server platform to the iSeries Integrated File System and execute it using the **java** command.

The Java Virtual Machine on iSeries server includes optimization capabilities that can improve the performance of Java code. The two main features are:

- ▶ Java Transformer
- ▶ Just-in-time (JIT) compiler

Both are controlled through Java Virtual Machine properties.

The Java Transformer is an iSeries server unique feature that precompiles Java classes and jar or zip files into *persistent, hidden* Java programs on the iSeries server. When you run a Java program with the **java** command, the system checks for the existence of the Java program and executes it if it is present. This is also known as *direct execution (DE)*.

A just-in-time compiler is a platform-specific compiler that generates *non-persistent* machine instructions for each method *upon the first call* to that method. These machine instructions exist only as long as the JVM that created them runs. You may notice a slower startup of the Java Virtual Machine since the JIT is compiling many methods for the first time. After this initial slow startup, the user program eventually achieves a state where most of the necessary methods have been compiled and the application runs close to direct execution mode.

You can run your Java programs in DE mode or JIT mode. Since V4R5 of OS/400, running your Java program *by default* results in the invocation of the JIT compiler, which would compile your bytecode to machine instructions and execute the compiled code. Before V4R5, the default was using the Java Transformer. The following section shows how to change between JIT and DE mode.

Under OS/400 V5R1, it is possible to perform DE optimizations across an entire set of jar files, provided these files are stored in the same subdirectory. This provides a very high degree of binding between the classes and results in substantial performance improvements. This technique is known as *Whole Program Optimization (WPO)*.

## 2.2.1 Creating a Java program

To create a Java program, you execute the CRTJVAPGM command. You can invoke the command against a single class, a group of classes (using the \* wildcard), a jar/zip file, or a group of jar/zip files. The CRTJVPGM command gives you an option as to the level of optimization you want to use. These options are:

- ▶ **\*INTERPRET:** The Java programs that you create are not optimized. When you start the program, it interprets the class file bytecode. You can display and change variables while debugging.
- ▶ **10:** The Java program contains a transformed version of the class file bytecodes, but has only minimal additional compiler optimization. You can display and change variables while debugging. This is the default value for the OPTIMIZE parameter.
- ▶ **20:** The Java program contains a compiled version of the class file bytecodes and performs additional compiler optimization. You can display variables, but not change them while debugging.
- ▶ **30:** The Java program contains a compiled version of the class file bytecodes and has more compiler optimization than optimization level 20. You can display, but not change, variables while debugging. The values that are presented may not be the current value of the variable.
- ▶ **40:** The Java program contains a compiled version of the class file bytecodes and performs more compiler optimization than optimization level 30. All call and instruction tracing is disabled.

**Note** If your Java program fails to optimize or throws an exception at optimization level 40, use optimization level 30.

In addition, the CRTJVAPGM command has a parameter called ENBPFRCOL. This allows you to specify whether performance data should be collected. Make sure you choose the proper value if you want to analyze the performance of your Java application. The default value of \*NONE disables performance data collection for that class or set of classes.

The possible values for the ENBPFRCOL parameter are:

- ▶ **\*NONE:** The collection of performance data is not enabled. No performance data is to be collected. This is the default for the ENBPFRCOL parameter.
- ▶ **\*ENTRYEXIT:** Performance data is collected for procedure entry and exit.
- ▶ **\*FULL:** Performance data is collected for procedure entry and exit. Performance data is also collected before and after calls to external procedures.



**Important:** Do not attempt to run the CRTJVAPGM command against the java.zip, sun.zip, or rt.jar files supplied by IBM on your iSeries. If you do, they will become corrupted and you will need to re-install the JV1 product to recover.

You can run CRTJVAPGM against the classes used by the native JDBC driver (part of JV1), the Java Toolbox classes installed with JC1, and the open version of the Java Toolbox (JTOpen).

## 2.2.2 Difference between direct execution and JIT

The JIT compiler is a platform-specific compiler that generates machine instructions for each method upon the first call to that method. You may notice a slower startup of the Java Virtual Machine because the JIT is compiling many methods for the first time. After this initial slow startup, the user program will eventually achieve a state where most of the necessary methods have been compiled and the application will run at close to direct execution optimization level 30 speed.

To understand the difference between the JIT compiler and direct execution, see the comparison in Table 2-1.

Table 2-1 Comparison of direct execution mode with JIT compilation

Characteristic	Direct execution	JIT Compiler
Execution speed	Faster for large programs	Faster for small programs
Loading speed	Faster if the code is pre-compiled; slower if is not pre-compiled	Faster if methods are called more than once; slower if most methods are used a small number of times
Memory use	Less if many processes are using the same program	Equal to static compilation if one or only a few processes are using the same program
Disk use	Uses four to five times the space of JIT	Uses four to five times less space than static compilation
Ease of use	More user management	Invisible to user
Binding	Tight binding within a JAR, loose binding between classes or between different JAR files	Tight binding for all classes and JAR files

## 2.3 Java garbage collection

*Garbage collection* is the process of freeing the storage used by objects that are no longer referred to by any program. Because of garbage collection, Java programmers do not have to write error-prone code to explicitly free (or delete) their objects that can result in "memory leak" program errors. The garbage collector automatically detects an object or group of objects that a user program can no longer reach, since there are no references to that object in any program structure. Once the object is collected, JVM automatically allocates the storage space for other uses.

Garbage collection on most platforms other than the iSeries uses a methodology known as “Stop and Copy”. This means that while the garbage collection takes place, all threads are stopped and the garbage collector takes over control of the entire memory to clean it up. While this is quite efficient, it has the disadvantage that it can cause unpredictable pauses in the users response time.

The iSeries, on the other hand, has concurrent garbage collection, which results in consistent performance to the end user. This is a major advantage in a business environment.

### **2.3.1 Understanding garbage collection**

Garbage collection has a price. Whether it is running as “Stop and Copy” or concurrent, it must still use CPU resources to perform its task. Understanding how the garbage collector works can help you understand how to tune it.

The garbage collector execution is based on the initial heap size parameter. You can set this parameter in the Java command line options.

The initial heap size parameter is a threshold that triggers a garbage collection cycle. It does not trigger a garbage collection cycle based on the absolute size of the heap, but on the amount the heap has grown since the last cycle (or since startup, if no cycle has yet been run). If you specify an initial heap size of 500 MB, the first cycle will run after 500 MB of space has been allocated. The garbage collector will then free some of that space for reuse. Then it will wait until another 500 MB is allocated before running again. If the garbage collector freed 170 MB in the first collection cycle, it won't run again until the heap size is at least 830 MB.

Under Java on the iSeries, the amount of memory allocated almost never shrinks. If you have a heap size of 500 MB and collect 170 MB, the heap size will in most cases still be 500 MB. Sometimes some heap space is recovered when deleting large objects, that is, objects larger than 8 KB. This space is recovered if a maximum heap size parameter is specified or if the number of large objects deleted in every GC cycle has been small. Even so, the heap space is unlikely to shrink much, because large objects usually take only a small fraction of the total allocated space.

The iSeries garbage collector does not perform heap compaction. The 170 MB of heap is unused until it is needed again by the JVM.

The initial heap size essentially determines the frequency of garbage collections. You can monitor the time interval between garbage collections. At times, increasing the garbage collection interval on a busy system can result in improved performance. This is usually best determined through trial and error.

**Note:** The iSeries definition of initial heap size is different from other platforms. On other platforms, the initial heap size determines the initial amount of memory to allocate, and the maximum heap size has more influence on when garbage collector runs. In most cases, the maximum heap size should not be specified on iSeries. This is different from other platforms. Cross-platform tuning tips may indicate that the max heap size should be set.

### 2.3.2 Problems with garbage collection

While garbage collection is supposed to relieve the programmer of the tedious task of freeing memory after use, it is not perfect. Since memory is only freed after all references to an object have been removed, you have the potential for memory leaks under certain circumstances. An example is when an object (although no longer needed) is kept in an object cache. Since the cache will retain the object reference for the life of the application, the object will never be garbage collected. Good coding practices can prevent such a scenario from occurring.

## 2.4 IBM Toolbox for Java

IBM Toolbox for Java is a set of Java classes that allow you to use Java programs to access data and resources on your iSeries and AS/400e servers. You can use these classes to write client/server applications, applets, and servlets that work with data on your iSeries. You can also run Java applications that use the IBM Toolbox for Java classes on the iSeries Java Virtual Machine.

IBM Toolbox for Java uses the iSeries Host Servers as access points to the system. Each server runs in a separate job on the server, and each server job sends and receives data streams on a socket connection. The IBM Toolbox for Java also provides the JDBC drivers used for interaction with DB2 on the iSeries server. Figure 2-3 provides an overview of jobs and services provided by the Toolbox.

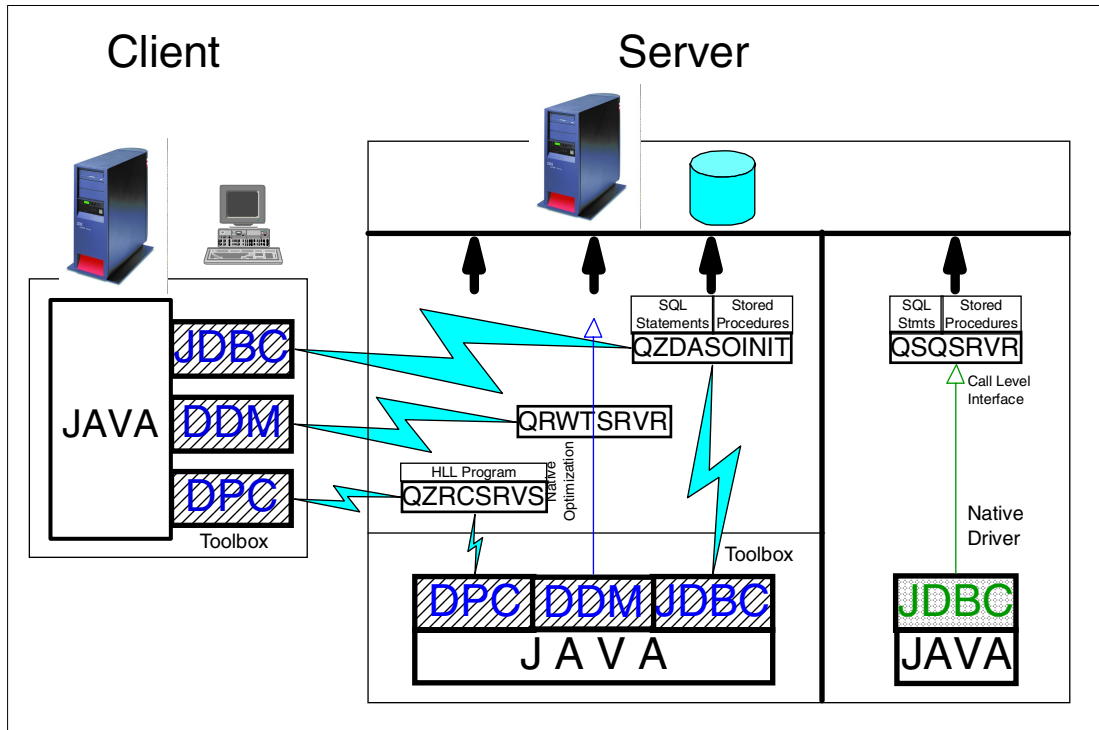


Figure 2-3 Accessing iSeries resources through IBM Toolbox for Java

The client (left-hand side of the picture) is running a Java program that accesses server resources in three different ways: using Java Database Connectivity (JDBC), Distributed Data Management (DDM), and Distributed Procedure Call (DPC). On the server side, JDBC access is handled within the QZDASOINIT job and DDM within the QRWTSRVR job, while DPC access is handled with the QZRCSRVS job.

From a performance perspective, it is especially important to understand the difference between using sockets (TCP/IP) and a native Call Level Interface for communication. While IBM Toolbox for Java provides a JDBC driver that uses *sockets*, the iSeries Developer Kit for Java provides a driver that uses *native calls*. The IBM Toolbox for Java driver is intended to be used by programs that access the database on a *separate system*. However, it will work also if the programs reside on the same system as the database. The native driver is intended and works only when programs run on the same iSeries server where the data is located. While the toolbox driver talks to the iSeries using TCP/IP via a QZDASOINIT job (regardless of whether the database is on the same or another system), the native JDBC driver talks directly to the database using a QSQSRVR job. For this reason, the native driver provides far superior performance.

If performance is important to your application and you have no plans to port the application to platforms other than the iSeries, you may also consider using some of the more advanced features of IBM Toolbox for Java (such as DDM and DPC). There are times where DDM can provide superior performance, especially for simple database access.

Chapter 6, "Java and WebSphere application design" on page 111, discusses the IBM Toolbox for Java in greater detail.



## Performance methodology and tools

This chapter introduces the basics of performance analysis methodology, as it relates to the iSeries WebSphere Application Server environment. You should be generally familiar with the practice of performance management since this is not a complete discussion on performance methodology. In a live production environment, a number of factors may influence the system performance, such as system tuning and proper system sizing.

This chapter begins by briefly reviewing the performance process, and then moves into those specific tools that have been chosen for application analysis. For each of the selected tools, we provide a concise description of how the tool is used and the specific advantages that each brings to the measurement process.

This chapter is not a substitute for reading the appropriate documentation for each tool, but contains sufficient detail that a person familiar with the tool will see how to apply it to the problem at hand.

## 3.1 Setting performance objectives

The first step in managing system performance is to set measurable objectives. You can begin by setting goals that match the demands of your business and identifying areas of the system where performance improvements can have a positive effect.

The following tasks make up a performance strategy. Implementing a performance strategy is an iterative process that begins with defining your performance goals or objectives, and then repeating the rest of the tasks until you accomplish your goals.

1. Set performance goals.
  - Set goals that match the demands of your business.
  - Identify areas of the system where an improvement in performance can affect your business.
  - Set goals that can be measured.
  - Make the goals reasonable.
2. Collect performance data continuously.
  - Always save performance measurement data before you install a new software release, a major hardware upgrade, a new application, or a large number of additional users or jobs.
  - Your data should include typical medium to heavy workloads.
3. Check and analyze performance data.
  - Summarize the collected data and compare the data to objectives or resource guidelines.
  - Perform monthly trend analysis that includes at least the previous three months of summarized data. As time progresses, include at least six months of summary data to ensure that a trend is consistent. Make decisions based on at least three months of trend information and your knowledge of upcoming system demands.
  - Analyze performance data to catch situations before they become problems. Performance data can point out objectives that have not been met. Trend analysis shows you whether resource consumption is increasing significantly or performance objectives are approaching or exceeding guideline values.
4. Tune performance whenever guidelines are not met.

The changes you make in tuning your system should be one at a time. If you make slight changes, you can determine the effect of each specific change. By prioritizing the changes, you can continue making changes and measuring performance until you reach your objectives.

5. Plan for capacity when:
  - Trend analysis shows a significant growth in resource utilization.
  - A major new application, additional workload, or new users, will be added to the current configuration.
  - You have reviewed business plans and expect a significant change.

Capacity planning and sizing are covered in more detail in Chapter 9, “Sizing and capacity planning” on page 181.

## 3.2 General tuning process and performance strategy

We believe that the most effective tuning strategy must commence at the operating system level and proceed from there to more detailed tuning options. To achieve optimal performance and stability of a WebSphere or Java application, you should follow these steps:

1. Pre-requisite: Ensure adequate system resources.

Before you can even start thinking about tuning your system, you must be confident that you have a system that is up to the task at hand. Certain iSeries and AS/400e server models are inadequate for Java and WebSphere work. Older AS/400 models might have high CPW ratings, but when it comes to executing Java, CPU Mhz, and the size of L2, cache is more important. This is covered in detail in Chapter 9, “Sizing and capacity planning” on page 181.

2. Understand the environment.

The first step in the tuning process is to understand what you are tuning. Java applications behave differently than traditional RPG and Cobol applications. They are multi-threaded by nature, which breaks several of the traditional rules of thumb used for OS/400 tuning. The Java environment is covered in Chapter 2, “The Java execution environment on iSeries servers” on page 7.

3. Optimize OS/400 for your environment.

Once you understand how Java works on the iSeries server, you can start looking for tuning opportunities at the operating system level. Tuning at this level usually provides good gain with little pain, and sometime a simple setting change can have a big impact. You can learn more about tuning the OS/400 environment in Chapter 4, “Tuning iSeries for a WebSphere or Java environment” on page 57.

4. Tune the WebSphere Application Server.

The next relatively easy opportunity for performance gains is with the WebSphere Application Server itself. There are a large number of “knobs” you can turn to tweak that extra bit of performance from you system. These are covered in Chapter 5, “Tuning HTTP server and WebSphere Application Server” on page 83.

5. Analyze and tune application code.

The next, and often the most costly, step is to change the application code. Understanding the coding practices that adversely impact performance is important. Identifying where poor performing code is located is difficult. A set of best practices for Java and WebSphere development is presented in Chapter 6, “Java and WebSphere application design” on page 111. This chapter also provides an overview of some of the tools that can help you in this chapter.

6. Scale the application.

If all your tuning efforts fail, you can consider scaling the application by using clustering or simply by spreading the load over multiple WebSphere instances. This should generally be considered a last resort, because there is, in fact, no guarantee that your application will scale gracefully or is capable of handling such an environment. Scaling is covered in Chapter 8, “Scaling the WebSphere environment” on page 167.

7. Load or stress testing to prove the expected benefit.

This step is optional and beyond the scope of this redbook. However, we urge application developers to consider load or stress testing their application several times throughout the development cycle. Performance problems can in this way be identified as early as the prototype stage and corrective action to the code can be applied before bad practices

spread throughout the application. It is a step that can be relatively time consuming, but far cheaper (in terms of time, money, and customer satisfaction) than putting an application that has not been load-tested into production. Also, it can have a significant payback when a greater degree of confidence is required.

### 3.3 Performance problem determination process

We strongly suggest that a formal problem determination process be used, since this will increase a probability of making all necessary measurements. The objective should always be to resolve the problem with the minimum (measurement) effort. Such a process should roughly follow this path:

1. Initiation

It may sound trivial, but starting the process by defining the problem in a few words and ensuring agreement on the definition can save time and effort later.

2. Determine the area of focus

Again this may appear trivial. Consider all aspects of the problem, even those that seem unlikely and document the probabilities of being a causal factor, as well as the reasons supporting the initial conclusions.

In the event that later steps prove that an incorrect path was taken, it becomes easier to backtrack and validate prior decisions, or move easily into an alternate investigation path.

3. Exploration

This is the start of high-level measurements to ensure that the effort required for the detailed measurements remains justified. More information may be discovered, which may cause a re-evaluation of the earlier decisions.

4. Focus on specific areas

At this point, there should exist a relatively high degree of confidence that any detailed measurements taken will provide information required to resolve the situation.

5. Information collection

Frequently this is the most time consuming part of the investigation. It may be that the problem may only manifest itself in certain circumstances that are difficult to reproduce. Alternatively, it may be that the volume of data collected is large and the required detail is hidden in a mass of irrelevant information.

6. Resolution and closure

On the basis that the process above (or something similar) has been followed, then almost by definition, this step requires minimal effort. This does not say that additional work is unnecessary. Rather it says that for the specific exercise considered, a resolution is documented, and the problem initially defined has been resolved.

Of course, as the result of the process, it may occur that other productive avenues of performance investigation, perhaps unrelated to the specific problem, will be generated.

The following sections examine the tools that are available within the iSeries environment that can assist in the measurements to support the above process.



## 3.4 Overview of iSeries performance measurement tools

Depending on the type of problem and the depth of measurement, you can use a number of different tools. Of course, other tools are available, but we found that the tools shown here cover most issues that you are likely to encounter. Table 3-1 shows the tools we chose for the process.

Table 3-1 Performance analysis tools

Phase	Data collected by	Data analyzed by
System analysis	1. Management Central 2. Collection Services 3. WRKSYSSTS command 4. WRKSYSACT command 5. WRKACTJOB command	1. Management Central 2. Performance Tools for iSeries 3. Manually 4. Manually 5. Manually
WebSphere analysis	1. WebSphere Administration Instance	1. WebSphere Resource Analyzer
Java/application analysis	1. DMPJVM command 2. Performance Explorer in Trace Mode 3. DBMonitor 4. SQL Visual Explain	1. Manually 2. PTDV  3. Manually 4. Manually

### 3.4.1 Collecting performance data

You must collect performance data if you are planning to use the Performance Tools for iSeries to analyze the system performance. There are two ways to collect the performance data:

- From a 5250 (“green-screen”) display
- Operations Navigator

#### Collecting performance data from a 5250 display

To collect performance data from a 5250 display, you follow this process:

1. Enter the Start Performance Tools (STRPFRT) command. The IBM Performance Tools for AS/400 display appears as shown in Figure 3-1. Choose option 2 (Collect performance data).

PERFORM	IBM Performance Tools for AS/400	System: AS05
Select one of the following:		
1. Select type of status 2. Collect performance data 3. Print performance report 4. Capacity planning/modeling 5. Performance utilities 6. Configure and manage tools 7. Display performance data 8. System activity 9. Performance graphics 10. Advisor  70. Related commands		
Selection or command ==>		
F3=Exit   F4=Prompt   F9=Retrieve   F12=Cancel   F13=Information Assistant F16=System main menu		

Figure 3-1 Performance Tools for iSeries Main Menu

- The Collect Performance Data screen appears as shown in Figure 3-2. Choose option 1 (Start collecting data).

Collect Performance Data	AS05
	11/01/01 11:15:39
Collection Services status:	
Status . . . . .	: Stopped
Select one of the following:	
1. Start collecting data 2. Stop collecting data	
Selection or command ==>	
F3=Exit   F4=Prompt   F5=Refresh   F9=Retrieve   F12=Cancel	

Figure 3-2 Collect Performance Data display

- The Start Collecting Data screen (Figure 3-3) appears. Enter parameters for the collection and press Enter. To minimize the performance impact, we suggest that you do not create

the database files when collecting the data as indicated by **\*NO** in Figure 3-3. See the help text or *Performance Tools for iSeries*, SC41-5340, for a detailed explanation of the parameter values.

Start Collecting Data

Type choices, press Enter.

Library . . . . .

Collection interval (minutes) . . .

Retention period:

Days . . . . .

Hours . . . . .

Cycling:

Time to synchronize cycle . . . .

Frequency to cycle collections . .

Create database files . . . . .

Collection profile . . . . .

QMPGDATA

5.00

5

0

00:00:00

4

**\*NO**

\*STANDARDP

Name

0.25, 0.5, 1, 5, 15, 30, 60

\*PERM, 0-30

0-23

HH:MM:SS

1-24

\*YES, \*NO

\*MINIMUM, \*STANDARD, \*STANDARDP, \*ENHPCPLN

F3=Exit F12=Cancel

Figure 3-3 Start Collecting Data display

### Collecting performance data with Operations Navigator

To collect performance data with Operations Navigator, follow these steps:

1. Start Operations Navigator.
2. Expand the iSeries server for which you want to start the data collection.
3. Expand **Configuration and Service**.
4. Right-click **Collection Services**. Right-click and select **Start Collecting** as shown in Figure 3-4.

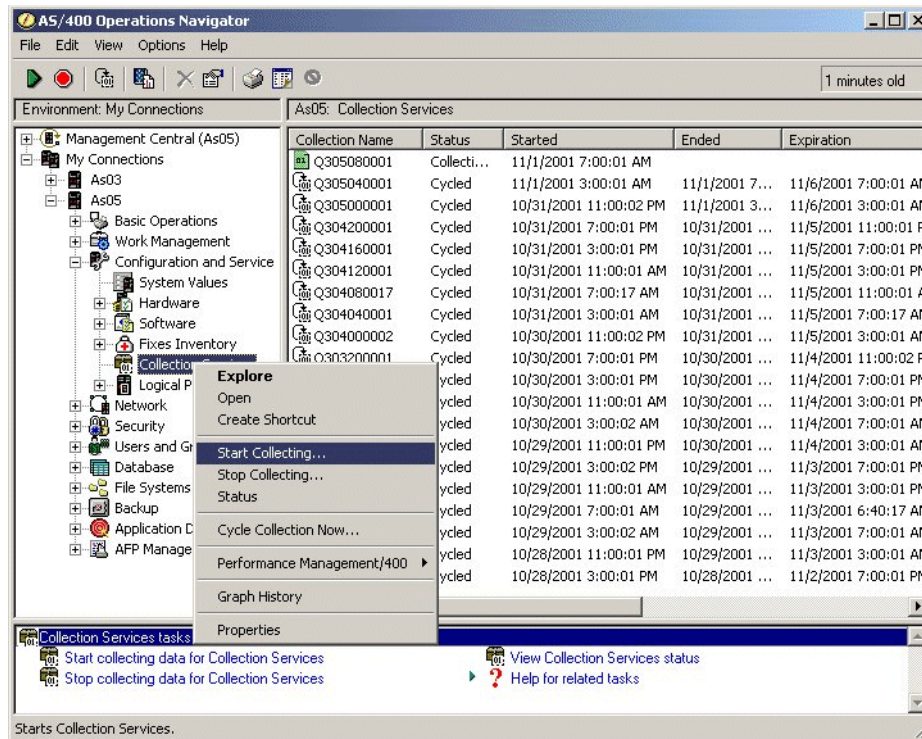


Figure 3-4 Starting Performance Data Collection with Operations Navigator

5. On the display that appears, specify your collection parameters and click **OK**.

### 3.4.2 Creating performance data

You must create the performance data files from the data collection object before you can use the Performance Tools for iSeries to analyze the data. You may choose to create the performance data while collecting it, but because of the system overhead, we suggest you collect the data and create the files only when needed.

#### Creating performance data from a 5250 display

To create performance data from a 5250 display, follow these steps:

1. Enter the Create Performance Data (CRTPFRTA) command. Press F4 and you see the Create Performance Data screen (Figure 3-5).

Create Performance Data (CRTPFRDTA)		
Type choices, press Enter.		
From collection . . . . .	_____	Name, *ACTIVE
Library . . . . .	QPFRRDATA	Name
To member . . . . .	*FROMMGTCOL	Name, *FROMMGTCOL
To library . . . . .	*FROMMGTCOL	Name, *FROMMGTCOL
Text 'description' . . . . .	*SAME	
Categories to process . . . . .	*FROMMGTCOL	Name, *FROMMGTCOL, *APPN...
+ for more values		
Time interval (in minutes) . . .	*FROMMGTCOL	*FROMMGTCOL, 0.25, 0.5, 1...
Starting date and time:		
Starting date . . . . .	*FROMMGTCOL	Date, *FROMMGTCOL
Starting time . . . . .		Time
Ending date and time:		
Ending date . . . . .	*FROMMGTCOL	Date, *FROMMGTCOL, *ACTIVE
Ending time . . . . .		Time
Bottom		
F3=Exit	F4=Prompt	F5=Refresh
F24=More keys	F12=Cancel	F13=How to use this display

Figure 3-5 Create Performance Data display

- Specify the values for the parameters and press Enter to create the database files.

## Creating performance data with Operations Navigator

Perform the following steps:

- Start Operations Navigator.
- Expand the iSeries server from which you collected the data.
- Expand **Configuration and Service**.
- Double-click **Collection Services**.
- Right-click your data collection, and select **Create Database Files Now** as shown in Figure 3-6.

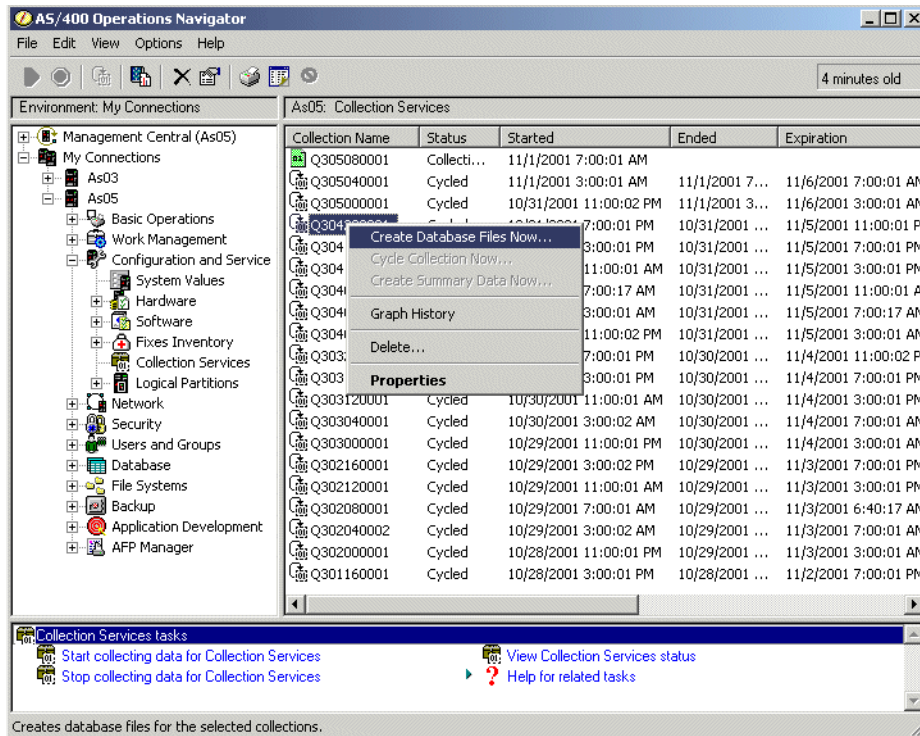


Figure 3-6 Creating the Performance Data with Operations Navigator

- The Create Database Files from window appears (Figure 3-7). Specify the library and the name for the database member and click **OK**.

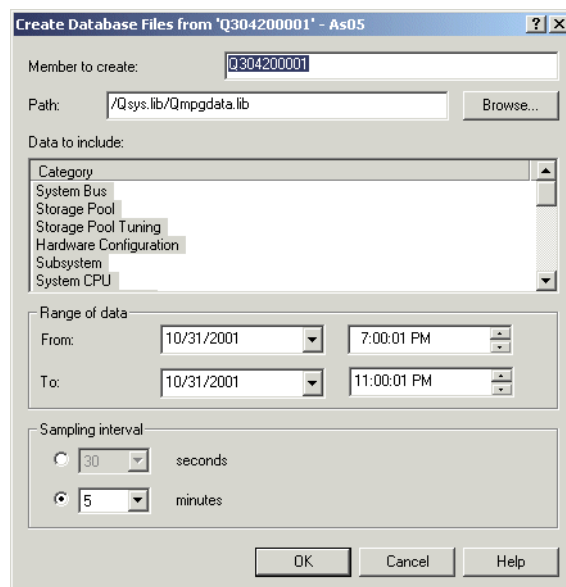


Figure 3-7 Create Database File dialog box

## 3.5 System level tools

System level tools refers to tools capable of measuring system resources, such as CPU usage, memory pool page faulting, disk utilization, and so on. The tools most often used for system wide performance monitoring are:

- ▶ Management Central
- ▶ WRKSYSSTS command
- ▶ WRKDSKSTS command
- ▶ WRKACTJOB command

One set of tools that you should not forget is System Service Tools (SST), which provide information about the hardware status on your iSeries server. Using the SST is not within the scope of this book. Refer to *AS/400e Diagnostic Tools for System Administrators*, SG24-8253, for detailed information about using these tools.

### 3.5.1 Management Central

Use Management Central's Collection Services to collect performance data as described in 3.4.1, "Collecting performance data" on page 19. You may use the Performance Tools for iSeries licensed program (5722-PT1) to create reports from the data. To collect and store performance data for future analysis, you can:

- ▶ Start Collection Services on a single system
- ▶ Start Collection Services on system groups
- ▶ Start Collection Services automatically

Collection Services collects data that identifies the relative amount of system resource used by different areas of your system. When you collect and analyze this information on a regular basis, you help balance your resources better, which in turn, gives you the best performance from your system. You can customize your data collections so you collect only the data you want.

You can use database files generated from collection services with the Performance Tools for iSeries licensed program (5722-PT1) or other applications to produce performance reports.

To view real-time performance data, Management Central provides an easy-to-use graphical interface for monitoring system performance. Figure 3-8 shows an example of a Management Central CPU monitor.

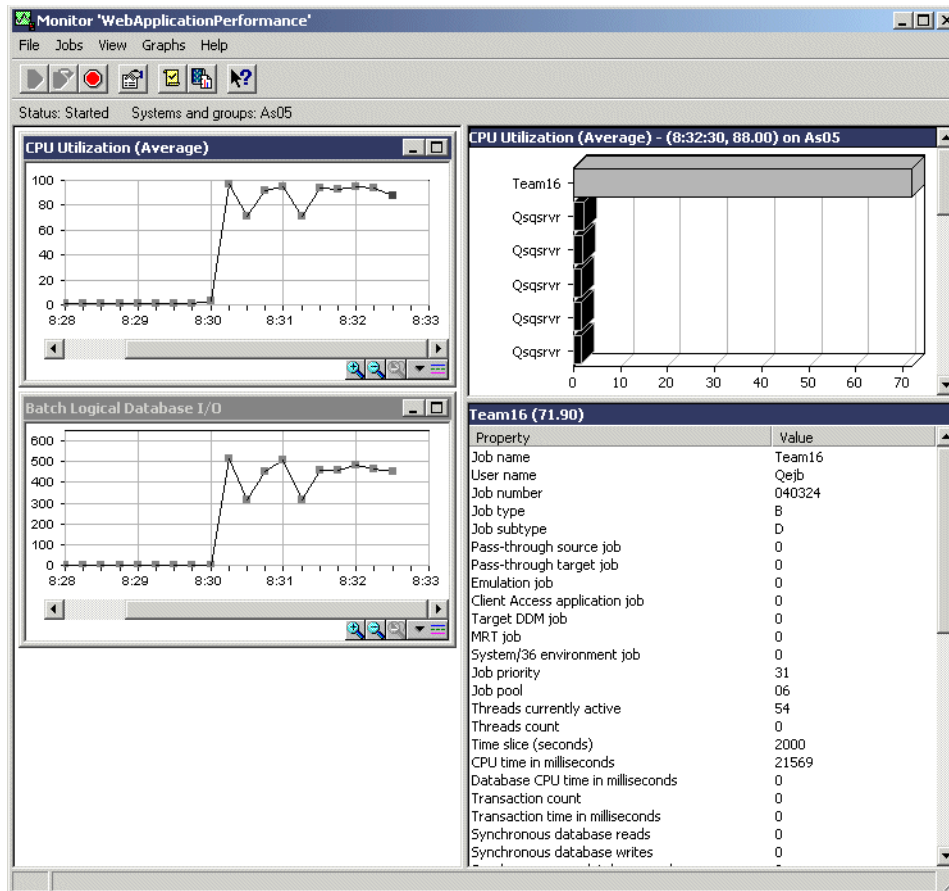


Figure 3-8 Sample Management Central view

In OS/400 V5R1, Management Central also added the capability to monitor individual jobs, groups of jobs, and servers such as WebSphere Application Server and HTTP server jobs.

### 3.5.2 WRKSYSSTS command

The Work with System Status display shows a group of statistics that depict the current status of the system. The Work with System Status display is actually two displays on one screen. The upper half shows statistics about the status of the system, while the lower half may be used to adjust the memory pool sizes and activity levels.

The upper half of the display provides you with the following information at a glance:

- ▶ Number of jobs currently in the system
- ▶ The total capacity of the system auxiliary storage pool (ASP)
- ▶ The percentage of the system ASP storage currently in use
- ▶ The amount of temporary storage currently in use
- ▶ The maximum amount of temporary storage space required since the last initial program load (IPL)
- ▶ The percentage of machine addresses used
- ▶ Statistical and reference information related to each storage pool that currently has main storage allocated to it

An example of a WRKSYSSTS display with the assistance level advanced is shown in Figure 3-9.



Work with System Status										AS05	
										10/12/01	10:12:35
% CPU used . . . . . :		97.8		System ASP . . . . . :				87.74 G			
% DB capability . . . . . :		12.5		% system ASP used . . . . . :				28.9425			
Elapsed time . . . . . :		00:00:03		Total aux stg . . . . . :				144.7 G			
Jobs in system . . . . . :		443		Current unprotect used . . . . . :				3135 M			
% perm addresses . . . . . :		.009		Maximum unprotect . . . . . :				3756 M			
% temp addresses . . . . . :		.015									
Sys	Pool	Reserved	Max	----	DB-----	--Non-DB---	Act-	Wait-	Act-		
Pool	Size M	Size M	Act	Fault	Pages	Fault	Pages	Wait	Inel	Inel	
1	493.68	200.95	+++++	.0	.0	.0	.0	.0	.0	.0	
2	2356.60	3.35	20	.0	.0	.0	.0	2990	.0	.0	
3	40.95	.00	5	.0	.0	.0	.0	.0	.0	.0	
4	504.75	.00	9	.0	.0	1.9	1.9	39.0	.0	.0	
5	200.00	.00	150	.0	.9	19.0	40.0	.0	.0	.0	
6	800.00	.99	200	.0	.0	10.9	125.0	13153	.0	.0	
										Bottom	
====>											
F21=Select assistance level											

Figure 3-9 WRKSYSSTS display

To change a pool size, enter a value in the field and press the Enter key. If the pool being changed is a private pool, this new pool size and the activity level shown in the Maximum Active column will both be used to change the subsystem description.

3.5.3 WRKDSKSTS command

Use the Work with Disk Status (WRKDSKSTS) command to observe how the disks are performing. An example of Work with Disk Status display is shown in Figure 3-10.

Work with Disk Status										AS05
										10/29/01 10:06:57
Elapsed time: 00:00:11										
Unit	Type	Size (M)	% Used	I/O Rqs	Request Size (K)	Read Rqs	Write Rqs	Read (K)	Write (K)	% Busy
1	6718	13161	21.5	.3	68.0	.0	.2	4.0	89.3	0
2	6718	13161	21.5	.0	128.0	.0	.0	128.0	.0	0
3	6718	13161	21.5	119.0	4.1	.0	118.9	4.0	4.1	4
4	6718	13161	21.5	37.2	4.1	.0	37.2	.0	4.1	0
5	6718	17548	21.5	118.6	4.0	.0	118.6	.0	4.0	4
6	6718	17548	21.5	77.5	4.0	.0	77.5	.0	4.0	8
7	6718	17548	21.5	37.9	4.9	.5	37.4	4.6	4.9	0
8	6718	13161	21.5	76.1	4.0	.0	76.1	.0	4.0	8
9	6718	13161	21.5	37.3	4.1	.0	37.2	4.0	4.1	0
10	6718	17548	21.5	77.8	4.6	.0	77.8	.0	4.6	0
11	6718	13161	21.5	78.0	4.0	.0	77.9	4.0	4.0	0
12	6718	13161	21.5	117.3	4.0	.0	117.2	4.0	4.0	4
										Bottom
Command										
===>										
F3=Exit F5=Refresh F12=Cancel F24=More keys										

Figure 3-10 WRKDSKSTS display

When viewing the Work with Disk Status display, observe the percent busy data. Each unit should be less than 50% busy. If each unit is between 50% and 70% busy, you may experience variable response times. If each unit is more than 70% busy, you might not have enough actuators (disk arms) to provide good performance. The actuator is the device within the disk unit that moves the read and write heads. If you have a well-tuned system with actuators that exceed 50% busy, you should increase the number of disk actuators.

If the percentage of usage varies noticeably (several per cents) from disk to disk, you should balance the disk unit data by running the Start ASP Balance (STRASPBAL) command. You should perform this operation every time after you add disk units to your iSeries server.

Pressing F11 on the display shown in Figure 3-10 provides you with information about the status of the individual disk units at a glance. An example of this is shown in Figure 3-11.

```

Work with Disk Status
AS05
10/29/01 10:06:57

Elapsed time: 00:00:00

--Protection--
Unit  ASP  Type  Status  Compression
  1    1  DPY   ACTIVE
  2    1  DPY   ACTIVE
  3    1  DPY   ACTIVE
  4    1  DPY   ACTIVE
  5    1  DPY   ACTIVE
  6    1  DPY   ACTIVE
  7    1  DPY   ACTIVE
  8    1  DPY   ACTIVE
  9    1  DPY   ACTIVE
 10    1  DPY   ACTIVE
 11    1  DPY   ACTIVE
 12    1  DPY   ACTIVE

Command
===>
F11=Display disk statistics  F16=Work with system status  F24=More keys

Bottom

```

Figure 3-11 WRKDSKSTS showing disk protection status

Contact your hardware service provider if the disk protection status is something other than *active*. A disk may have one of the following protection types:

- ▶ MRR for a mirrored disk
- ▶ DPY for a device in a device parity protection set

The disk protection status is one of the following types:

- ▶ **ACTIVE:** The type of protection specified in the type field is active for this unit. If the ASP is under system mirrored protection, mirrored protection is active for this unit.
- ▶ **BUSY:** Indicates that this unit is part of a disk unit subsystem that has device parity protection. This unit is not available for processing any commands.
- ▶ **DEGRADED:** Indicates that a decrease in performance has occurred because a component that is not critical has failed. No data is lost due to this failure, but the damaged unit needs to be repaired or replaced.
- ▶ **FAILED:** Indicates that a unit has failed. If another unit fails, data could be lost.
- ▶ **HDW FAIL:** Indicates that a hardware-related failure has occurred. The failure does not affect data or performance, but the probability of an outage caused by another failure of a redundant component (such as a power supply) is increased.
- ▶ **NOT RDY:** Indicates that this unit is part of a disk unit subsystem that has device parity protection. The unit is not ready to perform I/O operations.
- ▶ **PWR LOSS:** Indicates that this unit is part of a disk unit subsystem that has device parity protection. This unit has lost power.
- ▶ **REBUILD:** Indicates that the data on this storage unit is being rebuilt from another storage unit in the disk unit subsystem.
- ▶ **RESUME:** The unit is part of a mirrored ASP, and mirroring is in the process of being resumed on this unit.

- ▶ **RW PROT:** Indicates that this unit is part of a disk unit subsystem that has device parity protection. This unit is not accepting read or write operations.
- ▶ **SUSPEND:** The unit is part of a mirrored ASP, and mirroring is suspended on this unit.
- ▶ **UNKNOWN:** The protection status of the storage unit with device parity protection is not known to the system.
- ▶ **UNPROT:** The unit is operational, but another unit in the disk unit subsystem has failed or is being rebuilt.
- ▶ **WRT PROT:** Indicates that this unit is part of a disk unit subsystem that has device parity protection. This unit is not accepting write operations. Read operation is allowed.

### 3.5.4 WRKACTJOB command

The Work with Active Jobs display shows the performance and status information for jobs that are currently active on the system. All information is gathered on a job basis. The jobs are ordered on the basis of the subsystem in which they are running. By default, jobs that run in a subsystem are alphabetized by job name and indented under the subsystem monitor job field with which they are associated.

Press F11 key to toggle the display between various sets of information shown on the screen, and press F16 to arrange data according to any metrics you want. An example of this is shown in Figure 3-15 on page 32 where the WRKACTJOB display is arranged according to a number of threads per job. For example, perform the following steps when you want to see which job has the largest amount of threads running:

1. Enter the WRKACTJOB command.
2. Continue pressing F11 until the Threads column is displayed.
3. Move the cursor on to the Threads column.
4. Press F16 to rearrange the jobs according to the column.

The screens that you see in this process are shown in Figure 3-12 through Figure 3-15 on page 32.

Work with Active Jobs							AS05
							10/29/01 10:31:32
CPU %:	24.0	Elapsed time:	00:00:01	Active jobs:	320		
Opt	Subsystem/Job	User	Type	CPU %	Function	Status	
	QBATCH	QSYS	SBS	.0		DEQW	
	QCMN	QSYS	SBS	.0		DEQW	
	QCTL	QSYS	SBS	.0		DEQW	
	QSYSSCD	QPGMR	BCH	.0	PGM-QEZSCNEP	EVTW	
	QEJBSBS	QSYS	SBS	.0		DEQW	
	TEAM17	QEJB	BCI	.0	PGM-QEJBVR	JVAW	
	TEAM17ADMN	QEJB	BCI	5.9	PGM-QEJBADMIN	JVAW	
	TEAM17MNTR	QEJB	BCH	.0	PGM-QEJBMNTR	EVTW	
	QHTTSPVR	QSYS	SBS	.0		DEQW	
	ADMIN	QTMHHTTP	BCH	.0	PGM-QZHBHTTP	SIGW	
	ADMIN	QTMHHTTP	BCI	.0	PGM-QZSRHTTP	SIGW	
	DEFAULT	QTMHHTTP	BCH	.0	PGM-QZHBHTTP	CNDW	
	DEFAULT	QTMHHTTP	BCI	.0	PGM-QZHBHJOB	TIMW	
	DEFAULT	QTMHHTTP	BCI	.0	PGM-QZHBHJOB	TIMW	
	DEFAULT	QTMHHTTP	BCI	.0	PGM-QZHBHJOB	TIMW	
	DEFAULT	QTMHHTTP	BCI	.0	PGM-QZHBHJOB	TIMW	
							More...
====>							
F21=Display instructions/keys							

Figure 3-12 Basic WRKACTJOB display

Press F11 for the elapsed data display as shown in Figure 3-13.

Work with Active Jobs										AS05
										10/29/01 10:34:40
CPU %:	4.5	Elapsed time:	00:03:10	Active jobs:	321					
Opt	Subsystem/Job	Type	Pool	Pty	CPU	Int	Rsp	AuxIO	CPU %	
	QBATCH	SBS	2	0	.0			0	.0	
	QCMN	SBS	2	0	.0			0	.0	
	QCTL	SBS	2	0	.0			7	.0	
	DSP01	INT	2	10	.3	29	.0	80	.0	
	QSYSSCD	BCH	2	10	.0			0	.0	
	QEJBSBS	SBS	2	0	.0			0	.0	
	TEAM17	BCI	6	25+	8.1			0	1.1	
	TEAM17ADMN	BCI	6	25+	17.5			0	.6	
	TEAM17MNTR	BCH	6	25	.0			0	.0	
	QHTTSPVR	SBS	2	0	.0			0	.0	
	ADMIN	BCH	2	21	.6			0	.0	
	ADMIN	BCI	2	21	27.4			0	.0	
	DEFAULT	BCH	2	21	39.8			0	.0	
	DEFAULT	BCI	2	21	.0			0	.0	
	DEFAULT	BCI	2	21	.0			0	.0	
	DEFAULT	BCI	2	21	.0			0	.0	
										More...
====>										
F21=Display instructions/keys										

Figure 3-13 WRKACTJOB with elapsed data

Press F11 a second time, which provides you with the thread information screen shown in Figure 3-14.

Work with Active Jobs							AS05
							10/29/01 10:34:40
CPU %:	4.5	Elapsed time:	00:03:10	Active jobs:	321		
Opt	Subsystem/Job	User	Number	Type	CPU %	Threads	
	QBATCH	QSYS	051671	SBS	.0	1	
	QCMN	QSYS	051672	SBS	.0	1	
	QCTL	QSYS	051645	SBS	.0	1	
	DSP01	ACOVID	052709	INT	.0	1	
	QSYSSCD	QPGMR	051664	BCH	.0	1	
	QEJBSBS	QSYS	052518	SBS	.0	1	
	TEAM17	QEJB	052708	BCI	1.1	52	
	TEAM17ADMN	QEJB	052665	BCI	.6	98	
	TEAM17MNTR	QEJB	052663	BCH	.0	1	
	QHTTPSVR	QSYS	051833	SBS	.0	1	
	ADMIN	QTMHHTTP	051834	BCH	.0	1	
	ADMIN	QTMHHTTP	051872	BCI	.0	55	
	DEFAULT	QTMHHTTP	051836	BCH	.0	46	
	DEFAULT	QTMHHTTP	051840	BCI	.0	1	
	DEFAULT	QTMHHTTP	051843	BCI	.0	1	
	DEFAULT	QTMHHTTP	051845	BCI	.0	1	
							More...
====>							
F21=Display instructions/keys							

Figure 3-14 The WRKACTJOB with thread data

Move the cursor to the Threads column and press F16 to arrange the jobs on display according to the number of threads they have. This is shown in Figure 3-15.

Work with Active Jobs							AS05
							10/29/01 10:36:54
CPU %:	3.9	Elapsed time:	00:05:23	Active jobs:	321		
Opt	Subsystem/Job	User	Number	Type	CPU %	Threads	
	TEAM17ADMN	QEJB	052665	BCI	.4	98	
	ADMIN	QTMHHTTP	051872	BCI	.0	55	
	TEAM17	QEJB	052708	BCI	1.1	52	
	WPL01	QTMHHTTP	051908	BCH	.0	46	
	TEAM94	QTMHHTTP	051889	BCH	.0	46	
	TEAM92	QTMHHTTP	051883	BCH	.0	46	
	TEAM91	QTMHHTTP	051880	BCH	.0	46	
	TEAM90	QTMHHTTP	051870	BCH	.0	46	
	TEAM89	QTMHHTTP	051867	BCH	.0	46	
	TEAM88	QTMHHTTP	051864	BCH	.0	46	
	TEAM86	QTMHHTTP	051861	BCH	.0	46	
	TEAM41	QTMHHTTP	051856	BCH	.0	46	
	TEAM20	QTMHHTTP	051852	BCH	.0	46	
	TEAM17	QTMHHTTP	051839	BCH	.1	46	
	TEAM16	QTMHHTTP	051837	BCH	.0	46	
	DEFAULT	QTMHHTTP	051836	BCH	.0	46	
							More...
====>							
F21=Display instructions/keys							

Figure 3-15 WRKACTJOB sorted by the number of threads per job

Using the search facility on the WRKACTJOB display

You may also search for a string when viewing the WRKACTJOB display. Pressing F7 on any WRKACTJOB display provides you with the screen shown in Figure 3-16. Type in the string you are looking for, specify the column you want to find the string in, and press Enter.

Work with Active JobsAS05

Find a string

String . . . . .

RUN

Column . . . . .

\*STS

\*SBS, \*JOB, \*USER, \*NUMBER,  
\*TYPE, \*STS, \*FUNCTION, \*PTY,  
\*POOL

F8=Repeat find   F12=Cancel   F17=Top   F18=Bottom

TEAM20ADMN	QEJB	BCI	.0	PGM-QEJBADMIN	JVAV
TEAM20MNTR	QEJB	BCH	.0	PGM-QEJBMNTR	EVTW
QHTTPSVR	QSYS	SBS	.0		DEQW
ADMIN	QTMHHTTP	BCH	.0	PGM-QZHBHTTP	SIGW
ADMIN	QTMHHTTP	BCI	.0	PGM-QZSRHTTP	SIGW
ADMIN	QTMHHTTP	BCI	.0	PGM-QYUNLANG	TIMW
ADMIN	QTMHHTTP	BCI	.0	PGM-QYUNLANG	TIMW

More...

==>

F21=Display instructions/keys

Figure 3-16 WRKACTJOB with the ‘Find a string’ function

The screen shown in Figure 3-17 shows the result of searching for the “RUN” string in the job status column that we made in Figure 3-16.

Work with Active Jobs						AS05
						10/31/01 14:23:58
CPU %: 22.9		Elapsed time: 00:10:56		Active jobs: 341		
Opt	Subsystem/Job	User	Type	CPU %	Function	Status
	PETRIS	PETRI	INT	9.0	CMD-WRKACTJOB	RUN
	QPADEV0003	ACOVID	INT	8.0	CMD-WRKACTJOB	DSPW
	QSERVER	QSYS	SBS	.0		DEQW
	QPWFSEVSD	QUSER	BCH	.0		SELW
	QPWFSEVSO	QUSER	PJ	.0		DEQW
	QPWFSEVSO	QUSER	PJ	.0		DEQW
	QPWFSEVSO	QUSER	PJ	.0		DEQW
	QPWFSEVSO	QUSER	PJ	.0		DEQW
	QPWFSEVSO	QUSER	PJ	.0		DEQW
	QSERVER	QPGMR	ASJ	.0		EVTW
	QZDASVSD	QUSER	BCH	.0		SELW
	QZLSFILE	QUSER	PJ	.0		DEQW
	QZLSSERVER	QPGMR	BCH	.0		EVTW
	QSPL	QSYS	SBS	.0		DEQW
	PRTNP17	QSPLJOB	WTR	.0		EVTW
	PRTNP17	QSPLJOB	PDJ	.0		EVTW
===>						More...
F21=Display instructions/keys						

Figure 3-17 The WRKACTJOB display with Find String results

### 3.5.5 WRKSYSACT command

The Work with System Activity display allows the user to view performance data from a 5250 ("green screen") display in real-time fashion. This data is reported for any selected job or task that is currently active on the system. The performance statistics reported by this function represent activity that has occurred since a previous collection and they are reset on every collection interval. The major differences between WRKSYSACT and the other performance related commands are:

- ▶ WRKSYSACT is the only CL command that shows how both the jobs and tasks use the system resources.
- ▶ WRKSYSACT is a part of the Performance Tools for iSeries, 5722-PT1, Licensed Program Product.
- ▶ Only one user at a time may use the WRKSYSACT command.

An example of the WRKSYSACT display is shown in Figure 3-18.



Work with System Activity						AS05			
						10/16/01 08:37:51			
Automatic refresh in seconds . . . . .						5			
Elapsed time . . . . :		00:10:00		Average CPU util . . :		98.7			
Number of CPUs . . . :		2		Maximum CPU util . . :		99.0			
Overall DB CPU util :		8.8		Minimum CPU util . . :		98.4			
Type options, press Enter.									
1=Monitor job 5=Work with job									
						Total	Total	DB	
						CPU	Sync	Async	CPU
Opt	Task	User	Number	Thread	Pty	Util	I/O	I/O	Util
	TEAM16	QEJB	040324	000000B7	31	8.8	0	0	.0
	TEAM16	QEJB	040324	000000B9	31	8.6	0	0	.0
	TEAM16	QEJB	040324	000000B6	31	8.1	0	0	.0
	TEAM16	QEJB	040324	000000BB	31	7.8	0	0	.0
	TEAM16	QEJB	040324	000000BD	31	7.8	0	0	.0
	TEAM16	QEJB	040324	000000AF	31	7.8	0	0	.0
	TEAM16	QEJB	040324	000000BC	31	7.6	0	0	.0
	TEAM16	QEJB	040324	000000BE	31	7.6	0	0	.0
More...									
F3=Exit F10=Update list F11=View 2 F12=Cancel F19=Automatic refresh									
F24=More keys									

Figure 3-18 WRKSYSACT display

Press the F11 key to toggle between the four different views of this display in a rotating order. Press the F16 key to specify the metrics for how you want to order this display. The metrics used are:

- CPU percentage
- Amount of total disk I/O including both the synchronous and asynchronous I/O
- Net storage
- Allocated storage
- Deallocated storage
- Database CPU

### 3.5.6 Performance Tools for iSeries licensed program

The Performance Tools for iSeries program product includes tools for collecting data about the performance on a system, job, or a program level. It also includes tools to print reports and graphics out of the collected data to help you identify and correct performance related problems. Some commonly used features are outlined in the following sections.

#### CRTPFRTA command

Run the Create Performance Data (CRTPFRTA) command to create a set of database files from the performance information that is stored in a management collection (\*MGTCOL) object. The database files are discussed in detail in the iSeries Information Center (<http://publib.boulder.ibm.com/pubs/html/as400/v5r1/ic2924/index.htm>) under **Systems Management-> Performance**. The details of creating the database files are explained in 3.4.2, "Creating performance data" on page 22.

## DSPPFRDTA command

The Display Performance Data (DSPPFRDTA) command presents general information about the data collection and provides a summary of the performance statistics. You can use this command to analyze data that is collected from any iSeries server. Figure 3-19 shows an example of the Display Performance Data screen.

Display Performance Data			
Member . . . . .	TAKETHREE	F4 for list	
Library . . . . .	QPFREDBOOK		
Elapsed time . . . :	01:05:04	Version . . . . . :	5
System . . . . . :	AS05	Release . . . . . :	1.0
Start date . . . . :	10/25/01	Model . . . . . :	270
Start time . . . . :	20:00:02	Serial number . . :	10-12345
Partition ID . . . :	00	Feature Code . . . :	23F5-2434-1520
QPFRAJ . . . . . :	0	QDYNPTYSCD . . . . :	1
QDYNPTYADJ . . . . :	1		
CPU utilization (priority) . . . . . :	.00		
CPU utilization (other) . . . . . :	77.61		
Job count . . . . . :	222		
Transaction count . . . . . :	24		
Transactions per hour . . . . . :	22		
Average response (seconds) . . . . . :	.01		
Disk utilization (percent) . . . . . :	3.35		
Disk I/O per second . . . . . :	638.8		
Logical DB I/O for DDM jobs . . . . . :	0		
F3=Exit	F4=Prompt	F5=Refresh	F6=Display all jobs
F12=Cancel	F24=More keys	F10=Command entry	

Figure 3-19 DSPPFRDTA example

## Reports

Based on the data collected on the system, Performance Tools for iSeries provides you with two sets of reports. From the sample data, you can generate the following reports:

- ▶ System report
- ▶ Component report
- ▶ Job report
- ▶ Pool report
- ▶ Resource report

If you collected the trace data, you may also produce more detailed reports:

- ▶ Transaction report
- ▶ Lock report
- ▶ Batch job trace report

## Advisor

Advisor is a semi-automatic tuner that provides an easy-to-use way to improve many of the performance characteristics of your system. Advisor can help you to define specific tuning values and other parts of a processing environment to provide better performance for specific processing conditions on your system. Advisor analyzes performance data and then produces recommendations and conclusions to help improve performance. Advisor might recommend changes to basic system tuning values and may list conclusions about conditions that could cause performance problems.

You can choose to have the advisor change system tuning values as it recommends, or you can decide to make only the changes you select. You can use the Advisor's conclusions to:

- ▶ Make changes to your system
- ▶ Guide further performance data collection
- ▶ Help you request performance reports containing more information and explanations

Advisor may help you to improve system performance, but it does not identify or correct all performance problems for you. Performance information analyzed by Advisor includes:

- ▶ Storage pool sizes
- ▶ Activity levels
- ▶ Disk and CPU utilization
- ▶ Communications utilizations and error rates
- ▶ Input/output processor utilization
- ▶ Unusual job activities: Exceptions or excessive use of system resources
- ▶ Interactive trace data (when available)

Advisor does not:

- ▶ Make any recommendations for changing specific application programs to improve their performance
- ▶ Analyze non-interactive trace data

## 3.5.7 Using the system performance tools

If you suspect your system requires tuning changes, you need to investigate the items outlined in this section.

### CPU utilization

Java and WebSphere Application Server applications are typically CPU-intensive. Therefore, one of the first things to check is CPU utilization. Use the following commands to determine if there are too many jobs on the system or if some jobs are using a large percentage of processor time:

- ▶ Work with Active Jobs (WRKACTJOB): See examples in 3.5.4, "WRKACTJOB command" on page 30
- ▶ Work with System Activity (WRKSYSACT)

### Main storage

Use the Work with System Status (WRKSYSSTS) command to investigate faulting and the wait-to-ineligible transitions. See 4.3.4, "Setting the memory pool sizes and activity levels manually" on page 70, for details on using this command.

## Disk

Use the Work with Disk Status (WRKDSKSTS) command to determine if you have enough disk space and disk units. This is discussed in 3.5.3, “WRKDSKSTS command” on page 27.

## Communication lines

Use the following tools to find slow lines, errors on the line, or too many users for the line:

- ▶ Performance Tools for iSeries, Advisor
- ▶ Performance Tools for iSeries, Component Report
- ▶ Communications Trace: See *AS/400e Diagnostic Tools for System Administrators*, SG24-8253, for detailed information on using the trace

## IOPs

Use the following Performance Tools to determine if the workload between the IOPs is not evenly balanced or if there are not enough IOPs:

- ▶ Performance Tools for iSeries, Advisor
- ▶ Performance Tools for iSeries, Component Report

## Software

Use the Performance Tools for iSeries or the Work with Object Locks (WRKOBJLCK) command to investigate locks. You may also choose to use Performance Explorer, which is discussed in 3.7.2, “Performance Explorer (PEX)” on page 41.

## 3.6 WebSphere performance tools

WebSphere-level tools refer to tools capable of measuring the internal response time and resources used by the WebSphere Application Server. At present, the best tool available is the WebSphere Resource Analyzer.

### 3.6.1 WebSphere Application Server Resource Analyzer

WebSphere Application Server WebSphere Resource Analyzer is a performance monitor used for WebSphere Application Server *Advanced Edition*.

In WebSphere Application Server Version 3.5, Resource Analyzer is a free-of-charge, Technology Preview feature. It is available for download from the Web at:

[http://www.ibm.com/software/webservers/appserv/download\\_ra.html](http://www.ibm.com/software/webservers/appserv/download_ra.html)

In WebSphere Application Server Version 4.0, WebSphere Resource Analyzer is part of WebSphere Administrative Console. It is available as one of the options on the menu bar.

WebSphere Resource Analyzer retrieves performance data by periodically polling the administrative server. Data is collected continuously and retrieved as needed from within the Analyzer. The level of data to collect is specified by using the WebSphere Administrative Console. Then, the Analyzer's graphical interface is used to retrieve and view the data in a table or chart, or to store the data in a log file.

The Resource Analyzer provides a wide range of performance data for two kinds of resources:

- ▶ WebSphere Application Server resources such as enterprise beans and servlets
- ▶ Run time resources such as Java Virtual Machine memory, application server thread pools, and database connection pools

Performance data includes statistical data (such as the response time for each method invocation of an enterprise bean) and load data (such as the average size of a database connection pool during a specified time interval). This data is reported for individual resources and aggregated for multiple resources. Figure 3-20 shows a sample output from the WebSphere Resource Analyzer.

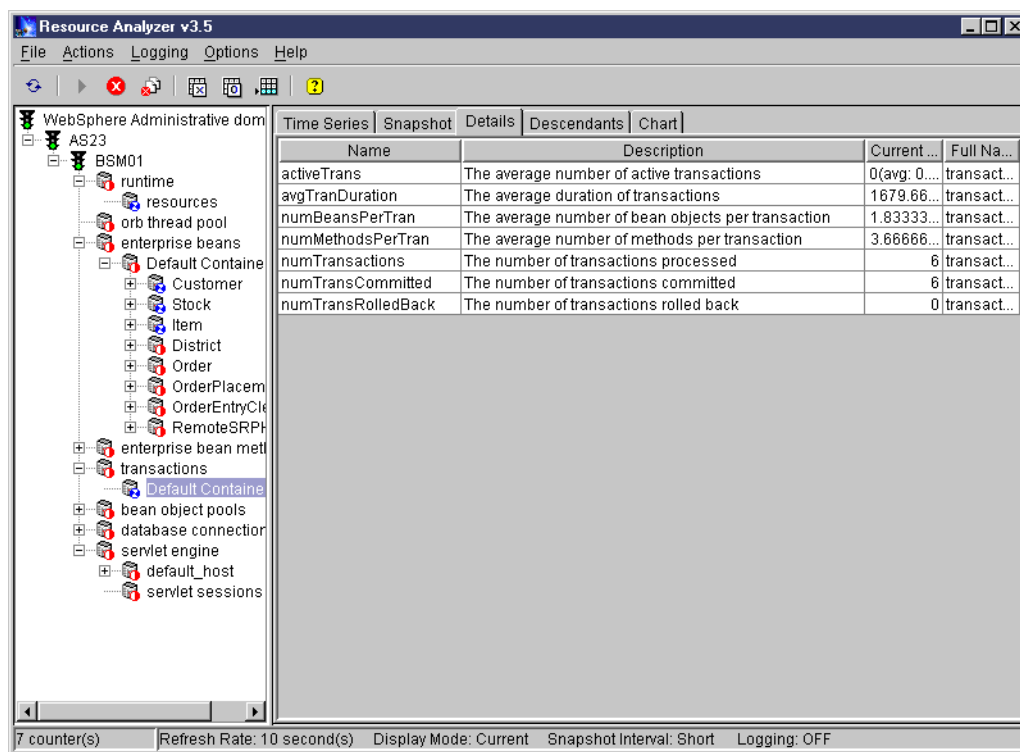


Figure 3-20 WebSphere Resource Analyzer

Depending on which aspects of performance are being measured, the Analyzer can be used to:

- ▶ View data in real time or during specified time intervals. For example, data can be displayed in intervals showing performance during the last minute, the last 5 minutes, the last 10 minutes, and the last 20 minutes.
- ▶ View data in chart form, allowing comparisons of one or more statistical values for a given resource on the same chart.
- ▶ Record current performance data in a log, and replay performance data from previous sessions.
- ▶ Compare data for a single resource to an aggregate (group) of resources on a single node.

### 3.7 Java and application-level tools

Application level tools refer to tools that are capable of looking “under the covers” of the running application. A number of tools are available, each with various levels of complexity. For a quick peek at what is going on in your JVM, use the Dump Java Virtual Machine (DMPJVM) command. For more in-depth analysis, use Performance Trace Data Visualizer (PTDV). And for database and SQL optimization, use DBMonitor and Visual Explain.

### 3.7.1 DMPJVM command

The Dump Java Virtual Machine (DMPJVM) command is a standard OS/400 command that dumps information about the Java Virtual Machine for a specified job. The information is produced with using printer file QSYSPRT. The user data for the QSYSPRT file is DMPJVM. The dump includes a formatted information about the classpath, garbage collection, and threads associated with the JVM as follows:

- ▶ The classpath
- ▶ Garbage collection
- ▶ Threads that are associated with the Java Virtual Machine

Figure 3-21 shows the classpath information output from the DMPJVM command. It includes information about the version of the JDK that was used.

```
Java Virtual Machine Information  034866/QEJB/WPL01
.....
. Classpath
.....
java.version=1.2
/QIBM/ProdData/Java400/jdk12/lib/jdkptf12.zip:/QIBM/ProdData/Java400/jdk12/
lib/rt.jar:/QIBM/ProdData/Java400/jdk12/lib/i18n.jar:/QIBM/ProdData/Java400
/ext/IBMmisc.jar:/QIBM/ProdData/Java400/ext/jssl.jar:/QIBM/ProdData/Java400
/ext/ibmjssl.jar:/QIBM/ProdData/Java400/:/qibm/proddata/http/public/jt400/l
ib/jt400.zip:/qibm/userdata/webasadv/WPL01:/qibm/userdata/webasadv/WPL01/Or
derEntryBeans.jar:/qibm/userdata/webasadv/WPL01/nonejb.jar:/QIBM/ProdData/J
ava400/ext/db2_classes.jar:/QIBM/ProdData/java400/ext/db2_classes.jar:/QIBM
/ProdData/WebASAdv/lib/wsa400.jar:/QIBM/UserData/WebASAdv/WPL01/properties:
/QIBM/ProdData/WebASAdv/lib/server/ibmwebas.jar:/QIBM/ProdData/WebASAdv/lib
/servlet.jar:/QIBM/ProdData/WebASAdv/lib/webt1srn.jar:/QIBM/ProdData/WebASA
dv/lib/server/ns.jar:/QIBM/ProdData/WebASAdv/lib/ejs.jar:/QIBM/ProdData/Web
ASAdv/lib/ujc.jar:/QIBM/ProdData/WebASAdv/lib/server/repository.jar:/QIBM/P
roddata/WebASAdv/lib/server/admin.jar:/QIBM/ProdData/WebASAdv/lib/was20cm.j
ar:/QIBM/ProdData/WebASAdv/lib/server/tasks.jar:/QIBM/ProdData/WebASAdv/lib
```

Figure 3-21 DMPJVM output: Classpath detail

Figure 3-22 shows the garbage collection information produced by DMPJVM. Using the DMPJVM command a few times while the application is running can give you information about the frequency of your garbage collections. It can also help you determine if you have memory leaks in your application code.

```
.....
. Garbage Collection
.....
Garbage collector parameters
  Initial size: 32768 K
  Max size: *NOMAX
Current values
  Heap size: 89376 K
  Garbage collections: 30
.....
```

Figure 3-22 DMPJVM output: Garbage collection detail

Figure 3-23 shows the Thread Information output of the DMPJVM command. You can see the call stack of any threads running as well their current status and any locking situations they may be in.

```
*...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+...
. Thread information
.....
Information for 49 thread(s) of 49 thread(s) processed
Thread: 00000013 Thread-0
  TDE: B000400006B9D000
  Thread priority: 5
  Thread status: Waiting
  Wait object: com/ibm/ejs/sm/server/ManagedServer
  Thread group: main
  Runnable: java/lang/Thread
  Stack:
    com/ibm/ejs/sm/server/ManagedServer.awaitShutdown()V+13 (ManagedServer.java:1046)
    com/ibm/ejs/sm/server/ManagedServer.main([Ljava/lang/String;)V+109
  (ManagedServer.java:141)
  Locks:
    None
.....
Thread: 00000043 Alarm Thread 1
.
.
```

Figure 3-23 DMPJVM output: Information for the first thread

The DMPJVM is an excellent tool for a quick overview of your application. If you are required to go deeper, the next tool called PTDV is probably a good step.

### 3.7.2 Performance Explorer (PEX)

PEX is a set of CL commands that come with OS/400. It is used for detailed performance data collection, which gives you the ability to analyze virtually every aspect of your running code and acquire profiling, statistical, and trace information for your needs. PEX allows you to collect performance data in three different modes:

- ▶ **Statistical:** Identifies applications and IBM programs or modules that consume excessive CPU use or that perform a high number of disk I/O operations. Typically, you use the statistical type to identify programs that should be investigated further as potential performance bottlenecks.
- ▶ **Profile:** Identifies high-level (HLL) programs that consume excessive CPU utilization based on source program statement numbers.
- ▶ **Trace:** Gathers a historical trace of performance activity generated by one or more jobs on the system. The trace type gathers very specific information about when and in what order events occurred. The trace type collects detailed program, Licensed Internal Code (LIC) task, OS/400 job, and object reference information. The amount of data gathered by a trace is enormous. That's why you would typically use trace after you narrow down a performance problem to a specific program or a job.

To define the mode as well as the details of your data collection, you follow these steps:

1. Create a PEX definition. The PEX definition is stored as a database member QAPEXDFN in library QUSRSYS.

2. Start PEX by using the Start PEX (STRPEX) command (“Collecting data for PTDV” on page 43 explains how to do this).
3. After you finish, end the PEX collection with End PEX (ENDPEX) command. PEX performance data is stored in files QAYPExxx in library QPEXDATA. The ENDPEX command places the collected data into members in each of the files in QPEXDATA.
4. To check your PEX definitions, use the command:

```
WRKMBRPDM FILE(QUSRSYS/QAPEXDFN)
```

5. Use the following command to check your data collections:

```
WRKMBRPDM FILE(QPEXDATA/QAYPExxx)
```

While PEX collects the performance data, it does not include any data analysis tool. You can analyze the data collected with PEX by using any of the following options:

- ▶ Using the Print PEX Report (PRTPEXRPT) command, which is a part of Performance Tools for iSeries. PRTPEXRPT takes all the information from the files in QPEXDATA, creates a readable report, and puts it into a spooled file.
- ▶ Creating your own queries against the files QAYPExxx in library QPEXDATA.
- ▶ Using the Performance Trace Data Visualizer (PTDV) tool, which is explained in 3.7.3, “Performance Trace Data Visualizer for iSeries (PTDV)” on page 42.

PEX is a tool that helps to find the causes of performance problems that cannot be identified by using other tools that are used for more general, system wide performance monitoring. It is designed for application developers who are interested in understanding or improving the performance of their programs. It is also useful for users knowledgeable in performance management to help identify and isolate complex performance problems.

You should use PEX after you have tried the other tools. It gathers vast amounts of data that can be analyzed to isolate the factors involved in a performance problem.

For more information on PEX, refer to *Performance Tools for iSeries* in the iSeries Information Center at: <http://as400bks.rochester.ibm.com/pubs/html/as400/v5r1/ic2924/index.htm>

### 3.7.3 Performance Trace Data Visualizer for iSeries (PTDV)

The IBM Performance Trace Data Visualizer for iSeries is a Java application that can be used for performance analysis of applications running on iSeries. PTDV works with the performance data collected by PEX trace mode. It allows you to view program flows and see details such as:

- ▶ CPU time
- ▶ Wall clock time
- ▶ Number of CPU cycles
- ▶ Number of instructions, which may be summarized by:
  - Trace
  - Job
  - Thread
  - Procedures
- ▶ Number of objects created
- ▶ Information about Java locking behavior

PTDV allows sorting and hiding of columns, exporting of data, and many levels of data summarization.



PTDV is available as a download from the IBM alphaWorks Web site at:  
<http://www.alphaworks.ibm.com>

PTDV ships with an excellent tutorial that describes all the features of the product and takes you through an example trace. We do not repeat the contents of this tutorial here, but we believe a brief overview of some of the features of the product is useful. For detailed information on performing some of the tasks described, please refer to the manual that comes with PTDV.

## Preparing to use PTDV

PTDV relies on event data being generated by the Java programs that are traced. Before you can collect data for PTDV, you must ensure that your Java programs are enabled for performance collection at least with the \*ENTRYEXIT option (see 2.2.1, “Creating a Java program” on page 10).

Alternatively, if you are using the JIT compiler, you must specify the following parameter at the command line of the JVM:

```
-Dos400.enbpfrcol=1
```

If you are using WebSphere Application Server, you specify this in the Administrative Console by clicking your Application Server Instance and editing command line arguments as shown in Figure 3-24.

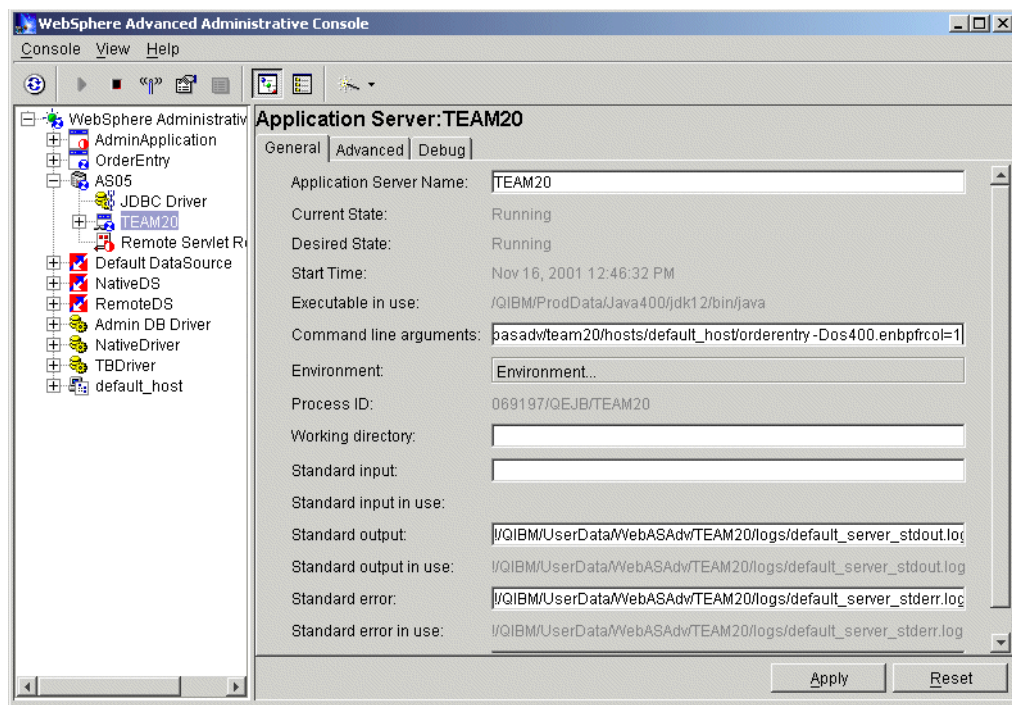


Figure 3-24 Enabling performance collection for JIT

Failure to use these options will result in no data showing up in PTDV.

## Collecting data for PTDV

Once your programs are prepared for performance collection and your application is running, you can start the PEX trace. The trace is started using the STRPEX command and ended using the ENDPEX command. You should keep the tracing period short because PEX collects a very large amount of data.

We suggest that you run only one transaction (if possible) when PEX is active. Even that one transaction may generate thousands of pages of data to be analyzed. During our case study shown in Chapter 7, “Case study” on page 127, we produced 9700 pages in a PEX report by running one transaction (an order entry) only.

The steps for using PEX and PTDV are:

1. Download the PTDV tool from the alphaWorks Web site (<http://www.alphaworks.ibm.com/>).
2. Install the PTDV on both the iSeries and the workstation following the PTDV installation instructions.
3. Prepare the application for performance collection.
4. Create a PEX definition using a definition provided with PTDV.
5. Start the PEX collection.
6. Run a small number of your application transactions, if possible, just one.
7. Stop the PEX collection.
8. Create reports from the databases or use PTDV.

Some of the above steps are explained in more detail in the following sections.

## Preparing the application for performance collection

A powerful feature of OS/400 is the ability to “pre-compile” and optimize Java classes into Machine Interface (MI) programs instead of using byte-code interpretation to execute the code. This is known as *direct execution (DE)*. OS/400 supports several levels of optimization ranging from \*INTERPRET (interpretation) to 40 (DE with good runtime optimization). The existence of a Java program is transparent to the Java programmer. If no Java program is explicitly created, JIT will be used. Java programs are created with the Create Java Program (CRTJVAPGM) command, which can be executed against both .class and .jar files.

You can display the status of your Java programs by using the Display Java Program (DSPJVAPGM) command:

1. At an OS/400 command line, type the following command (case sensitive):

```
DSPJVAPGM CLSF('/QIBM/UserData/WebASAdv/<Your Instance>/OrderEntryBeans.jar')
```

Here *OrderEntryBeans.jar* is a jar file that contains an application. The information about the jar file appears as shown in Figure 3-25.

```

                                Display Java Program Information
File name . . . . . : < ASAdv/TST01/OrderEntryBeans.jar
Owner . . . . . : BJARNE

Java program creation information:
File change date/time . . . . . : 03/19/01 10:10:17
Java program creation date/time . . . . . : 03/19/01 10:19:25
Java programs . . . . . : 94
Classes with current Java programs . . . . . : 94
Classes without current Java programs . . . . . : 16
Optimization . . . . . : *INTERPRET
Enable performance collection . . . . . : *NONE
Use adopted authority . . . . . : *NO
User profile . . . . . : *USER

```

Figure 3-25 Display Java Program Information

Notice that there is no optimization (\*INTERPRET) for this jar file and that performance collection is disabled (\*NONE).

2. Enable performance collection on the OrderEntryBeans.jar file.

At a command line, type the following commands:

```

CALL QCMD
SBMJOB CMD(CRTJVAPGM CLSF('/QIBM/UserData/WebASAdv/<Your Instance>
/OrderEntryBeans.jar') OPTIMIZE(40) ENBPFCOL(*ENTRYEXIT)) JOBQ(QBATCH)

```

**Using CRTJVAPGM:** The CRTJVAPGM is extremely CPU intensive and can take a long time to run. As a matter of principle, you should always run it in batch to take advantage of the higher CPW rating of the iSeries batch processing.

3. Use the DSPJVAPGM command again to confirm that your Java program creation was successful. The result should be similar to the example shown in Figure 3-26.

```

                                Display Java Program Information
File name . . . . . : < ASAdv/TST01/OrderEntryBeans.jar
Owner . . . . . : BJARNE

Java program creation information:
File change date/time . . . . . : 03/19/01 10:10:17
Java program creation date/time . . . . . : 03/19/01 10:19:25
Java programs . . . . . : 1
Classes with current Java programs . . . . . : 110
Classes without current Java programs . . . . . : 0
Optimization . . . . . : 40
Enable performance collection . . . . . : *ENTRYEXIT
Use adopted authority . . . . . : *NO
User profile . . . . . : *USER

```

Figure 3-26 Displaying an optimized jar file

4. Repeat step 2 for every jar file and class file that form part of the application. The remaining files are:

- /QIBM/UserData/WebASAdv/<Your Instance>/nonejb.jar
- /QIBM/UserData/WebASAdv/<Your Instance>/hosts/default\_host/OrderEntry/shopData/\*.class
- /QIBM/UserData/WebASAdv/<Your Instance>/hosts/default\_host/OrderEntry/Support/\*.class

Note that you do not need to do this for servlets. Servlets are always executed using JIT, because of the WebSphere class loader mechanism.

**Important:** Do not attempt to run the CRTJVAPGM command against the java.zip, sun.zip, or rt.jar files supplied by IBM on your iSeries. If you do, they will become corrupted and you will need to re-install the JV1 product to recover.

You can run the CRTJVAPGM command against the classes used by the native JDBC driver (part of JV1), the Java Toolbox classes installed with JC1, and the open version of the Java Toolbox (JTOpen).

5. Enable performance collection in WebSphere.

Since servlets are executed using the JIT, you must use a different method to enable performance collection. This is done through the WebSphere Administrator's Console.

- a. Click **Start->Programs->IBM WebSphere->Application Server V3.5->Administrator's Console**.
- b. When the console has completed loading, expand the tree down to your instance level.
- c. Append the following setting to the end of "Command line arguments" to enable performance collection (case sensitive):  
`-Dos400.enbpfrcol=1`
- d. Click **Apply**.

6. Restart the Application Server Instance.

- a. Right-click the Application Server Instance and click **Stop**.
- b. Wait for a success message dialog to appear.
- c. Click **OK** to close the success message dialog.
- d. Right-click the Application Server Instance and click **Start**.
- e. Wait for a success message dialog to appear.
- f. Click **OK** to close the success message dialog.

## Creating PEX definition for Java performance collection

You now create a definition for PEX that contains information about what you plan to measure. The PEX tool captures events sent by OS/400, and those events are subsequently used by PTDV to analyze the runtime behavior of the program. Since the number of events that it is possible to capture is extensive, it is important to select the right options.

1. From an OS/400 command line, type the Add PEX Definition command:

```
ADDPEXDFN
```

2. Press F4 to prompt. The command options appear as shown in the example in Figure 3-27.

Add PEX Definition (ADDPEDFN)		
Type choices, press Enter.		
Definition . . . . .		Name
Type . . . . .	*STATS	*STATS, *TRACE, *PROFILE
Job name . . . . .	*	Name, generic*, *ALL, *
User . . . . .		Name, generic*, *ALL
Number . . . . .		000001-999999, *ALL
	+ for more values	
Task name . . . . .	*NONE	
	+ for more values	

Figure 3-27 Adding a PEX definition

3. Press F11 to show the command keyword and then complete the options as follows:

- a. The first option is the name of your definition. It will be used when you start the PEX trace:

DFN(<Your Instance Name>)

- b. This option specifies that you want to run a trace (as opposed to statistics or profiling):

TYPE(\*TRACE)

- c. This option identifies the job you will trace. You can use a generic job name and user name.

JOB(Job name: <Your Instance>, User:QEJB, Number:\*ALL)

Since you will trace a specific WebSphere instance, you can narrow the field by specifying your instance name as the job name and the user QEJB, but a job number of \*ALL. This way you can use the same definition again and again, even after the instance job is restarted.

- d. This option specifies that you want to include trace data from the Licensed Internal Code:

TASK(\*ALL)

4. Press Enter. You are presented with additional options. Enter the following settings:

- a. This specifies the max size of the trace buffer in Kb. PEX traces can grow very large, very quickly. A size of 100 Mb, as specified here, should be sufficient for our test.

MAXSTG(100000)

- b. Specifies that you want to include specific, selected events in the trace. We specify the events later.

TRCTYPE(\*SLTEVT)

- c. If the trace type is \*SLTEVT, this option must be \*YES:

SLTEVT(\*YES)

5. Press Enter. You are presented with additional options that allow you to select the events you want to include. Enter the following settings:

- a. Performance Measurement Counter Overflow. This is a required parameter for PTDV. It instructs PEX to capture CPU usage events.

BASEVT(\*PMCO)

- b. Specifies that you want to trace Java program entry/exit events.

PGMEVT(\*JVAENTRY \*JVAEXIT)

- c. Specifies that you want to trace Java object creation events and thread state changes.

JVAEVT(\*OBJCRT \*THDSTTCHG)

6. Press Enter.

The PEX definition is added to the PEX repository. You are now ready to start a PEX trace.

## Collecting performance data for analysis

Before you start the performance collection, go through a practice run by running your application transaction once. This ensures that all one-time caching and initial class loading has taken place (we don't really want to include that in our trace). Then run the trace by following these steps:

1. Start PEX by issuing the following command:

STRPEX SSNID(<Your Instance Name>) DFN(<Your Instance Name>)

2. Run your application transaction.

3. Stop PEX by issuing the following command:

ENDPEX SSNID(<Your Instance Name>) RPLDTA(\*YES)

## Visualizing performance data with PTDV

Here is an example of using PTDV to analyze the result of a PEX trace. When you start PTDV, the trace is downloaded. After completing the download, the PTDV main panel appears as shown in Figure 3-28.

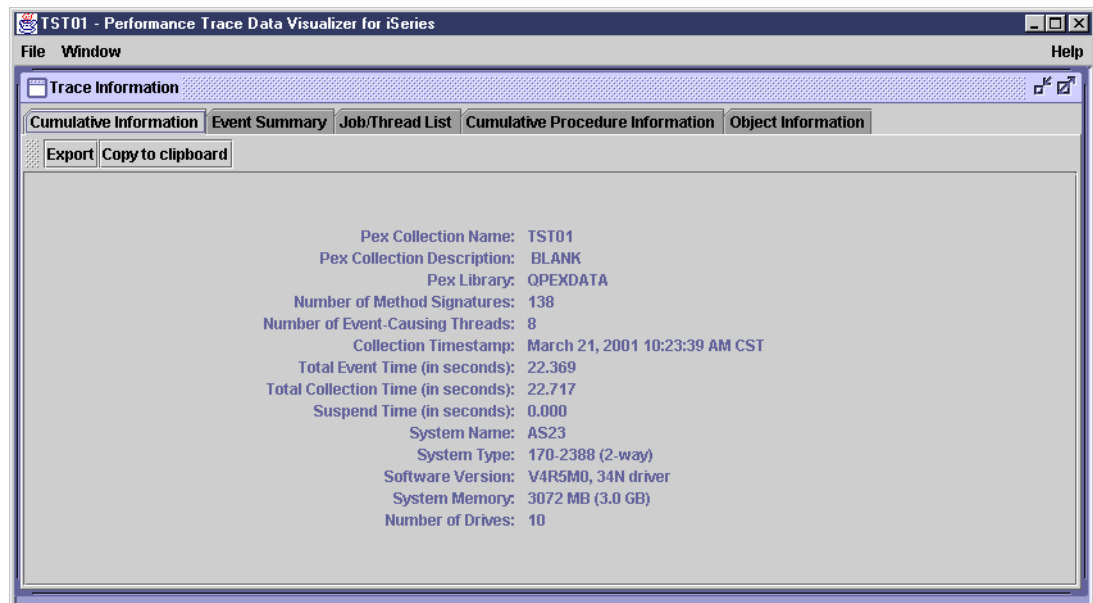


Figure 3-28 Initial summary tab: Cumulative Information

If the downloading of the trace data seems to hang, check the QSYSOPR message queue for the message CPA4072. If you receive the message, verify that the MAXRCDS setting of the printer file QPRINT is set to \*NOMAX. The starting of PTDV actually creates a spooled file out of the previously collected PEX trace data, and analyzing of the data only starts when the spooled file is complete. In case you are tracing either a poorly coded application or a system with a full workload, you will create a spooled file with thousands of pages.

PTDV will show many frames containing various information. You can identify the frames by the title, in this case, "Trace Information". PTDV allows you to move, resize, maximize, and minimize the frames that it shows.

Like most PTDV frames, the Trace Information frame has several tabs across the top. These tabs divide the information into manageable pieces. By clicking the various tabs, different panes are shown with information related to the name of the tab you clicked.

For the Trace Information frame, the tabs (panes) are:

- ▶ Cumulative Information
- ▶ Event Summary
- ▶ Job/Thread List
- ▶ Cumulative Procedure Information
- ▶ Object Information

### ***Cumulative Information***

The Cumulative Information is shown in Figure 3-28. You find basic information about the PEX collection and the system on which it was collected.

### ***Event Summary***

The Event Summary pane shows the number of events of each type contained in the trace. The events are subdivided according to their PEX types ("Java Events" and "Machine Interface Program Bracketing Events") and subtypes ("Java Object Create/Delete", "Java Entry", and "Java Exit"). The top line contains the total number of events.

You will notice that the table has two columns of event counts: "Events in Trace" and "Events Processed". Most of the time, these two columns are identical. However, if you click the Stop button while PTDV is processing events, the numbers in the "Events Processed" column will be lower than the "Events in Trace", because not all events were processed.

### ***Job/Thread List***

From the Job/Thread List pane, you can see all of the jobs that were active during your PEX collection and the threads in those jobs. Figure 3-29 shows an example. The jobs are divided into two categories: "Event-causing jobs" and "Non-event-causing jobs". The event-causing jobs are the ones you are most likely interested in; they are the ones that caused PEX events. However, it is sometimes useful to look at the non-event-causing jobs to make sure that nothing unexpected was going on with the rest of the system while your program was running. In our sample, there are no non-event-causing jobs, because we performed the trace on a running Application Server Instance.

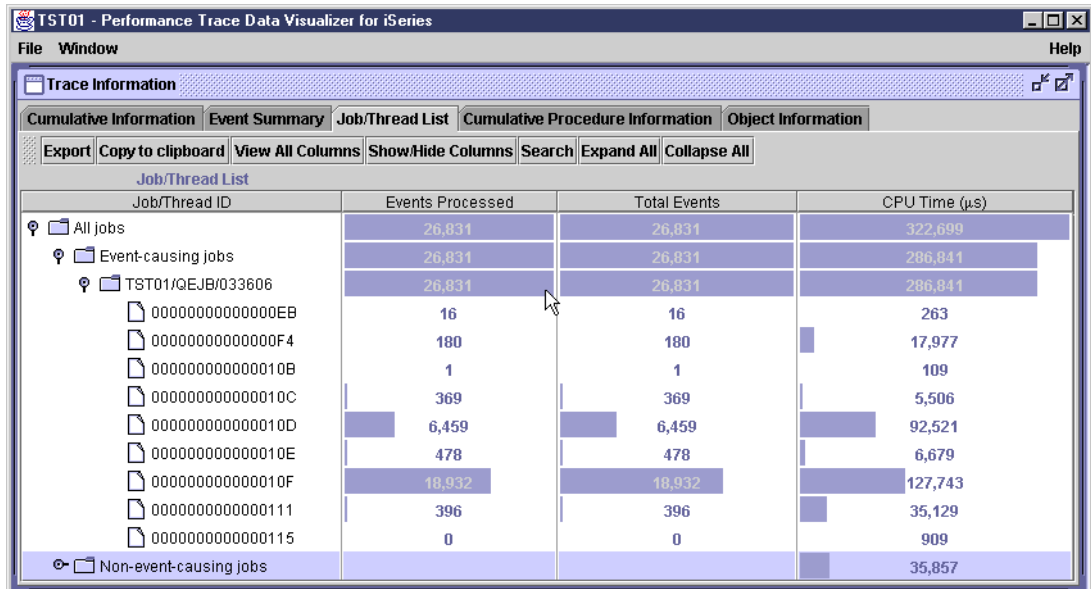


Figure 3-29 Job/Thread List

To the right of the job and thread names, you can see information about the job or thread. This is typical of most PTDV panes. While the information shown in the columns, by default, is usually the most useful, other information is available. You can click the View All Columns button to quickly display all of the available columns.

You can also right-click a column in the table header, and from the menu, select the Hide Column option.

If you click Show/Hide Columns at the top of the pane, you see a list of the available columns as shown in Figure 3-30. You can select which of those columns you want to view. This is often the easiest method to show only the information you are interested in.

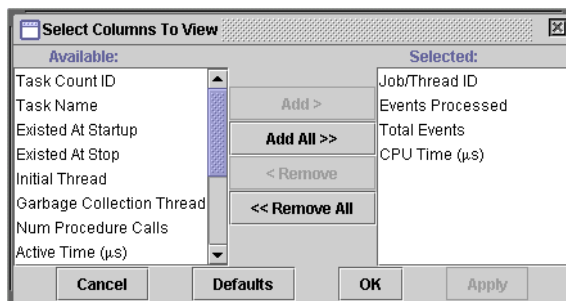


Figure 3-30 Selecting columns

### Object Information

The object information pane shows you information about the objects created during the trace. This includes the number of objects of each type that were created and the size of those objects.

### Cumulative Procedure Information

The cumulative procedure information pane contains summary information about all of the procedures/methods called during the trace, as shown in Figure 3-31.



Procedure Name	# Invocations	Inline CPU ...	Cumulative ...	Inline Objec...	Cumulative ...
<unknown>. <unknown>	8	62,856	286,351	774	26,291
DEMODO.com-ibm-itso-roch-cpwejb-interfaces-Address-<init>(Ljava-lang-S...	2	2	2	0	0
DEMODO.com-ibm-itso-roch-cpwejb-interfaces-OrderDetail-<init>(Ljava-lan...	1	2	2	0	0
DEMODO.com-ibm-itso-roch-cpwejb-interfaces-OrderDetail-getItemAmount(...	3	3	3	0	0
DEMODO.com-ibm-itso-roch-cpwejb-interfaces-OrderDetail-getItemID(Ljav...	3	2	2	0	0
DEMODO.com-ibm-itso-roch-cpwejb-interfaces-OrderDetail-getItemQty(I	3	1	1	0	0
DEMODO.com-ibm-itso-roch-cpwejb-interfaces-OrderDetail-getLineNumber...	1	0	0	0	0
DEMODO.com-ibm-itso-roch-cpwejb-interfaces-OrderDetail-setLineNumber...	1	0	0	0	0
DEMODO.com-ibm-itso-roch-wasaejb-CustomerBean-<init>()V	1	0	0	0	0
DEMODO.com-ibm-itso-roch-wasaejb-CustomerBean-ejbActivate()V	1	0	0	0	0
DEMODO.com-ibm-itso-roch-wasaejb-CustomerBean-ejbFindByPrimaryKey...	1	1	6,683	0	494
DEMODO.com-ibm-itso-roch-wasaejb-CustomerBean-ejbLoad()V	1	2,997	10,135	254	748
DEMODO.com-ibm-itso-roch-wasaejb-CustomerBean-ejbPassivate()V	1	0	0	0	0
DEMODO.com-ibm-itso-roch-wasaejb-CustomerBean-ejbStore()V	1	1,877	1,877	244	244
DEMODO.com-ibm-itso-roch-wasaejb-CustomerBean-getEntityContext()Ljava	1	2	2	0	0

Figure 3-31 Cumulative Procedure Information

The "Procedure Name" column contains the names of all the methods called. The first part of the name (DEMODO) tells you how the iSeries executed the method. DEMODO means "direct execution mode" (running an optimized Java program), and JITC indicates the just-in-time compiler. Calls to JNI functions may include other modules.

You will notice that in several cases, there are "inline" and "cumulative" versions of a column. For example, there is "Inline CPU Time (µs)" and "Cumulative CPU time (µs)". In these cases, *inline* refers to the time spent in this method only, while *cumulative* refers to the time spent in this method and all of the methods called by this method.

You can click on any of the column headings to sort the output in that column. This provides a good entry point to determine which methods are the most like performance bottlenecks.

In addition, PTDV provides drill-down capabilities for most columns. Please refer to the PTDV Tutorial for an extensive description. You can locate the tutorial by following these steps:

1. Install PTDV.
2. Open Windows Explorer and click <ptdv install directory>\doc\index.html.
3. Click **Tutorial**.

### 3.7.4 Database Monitor for iSeries

Database Monitor for iSeries (DB Monitor) is a tool that can be used to analyze database performance problems after SQL requests have completed running. While DB Monitor doesn't measure performance of WebSphere Application Server or Java programs directly, these programs often use SQL to access and manipulate the data, which may have a significant performance impact. The DB Monitor tool has been part of OS/400 since V3R6.

The iSeries performance analyst will use DB Monitor to gather database and performance data generated when SQL queries are executed. Then using customized SQL programs, the analyst can view, analyze, and conclude the most appropriate actions to take in order to generate the most efficient SQL queries possible for their application.

DB Monitor data is most useful if the user has a basic knowledge of iSeries query optimization techniques.

### Collecting DB Monitor data

The Start Database Monitor (STRDBMON) command starts the collection of database performance statistics for a specified job or all jobs on the system. The statistics are placed in a specified database file and member. If the file or member does not exist, it is created based on the QAQQDBMN file in library QSYS. If the file or member exists, the record format of the file is checked to see if it is the same.

Enter the STRDBMON command to start the DB Monitor. The DB Monitor collects information on previously started jobs or new jobs started after the monitor collection has begun. Because of the volume of data collected, try to gather data for a specific job only. This makes analysis easier and keeps the DB Monitor file smaller. If this is not possible, collect data on all of the jobs, and use queries to select the specific jobs of interest.

Please note that when the DB Monitor is gathering data, a significant amount of CPU utilization (20 to 30 percent) and disk usage may be temporarily required.

### Start Database Monitor (STRDBMON) parameters

The STRDBMON parameters and how they are used are listed here:

- ▶ **OUTFILE:** The file name for the results file is required, but the library name is optional. The file will be created if it does not exist; it will be reused if it exists.
- ▶ **OUTMBR:** This parameter defaults to the first member in the file. Specify the “ADD” or “REPLACE” option (the default is “REPLACE”). The “ADD” option causes the new results to be appended to the end of the file.
- ▶ **JOB:** This parameter defaults to the job issuing the STRDBMON command. The user can specify one job or \*ALL jobs – no subsetting is allowed. Two DB Monitors can collect data on the same job.
- ▶ **TYPE:** This parameter allows the user to specify the type of data to be collected – \*SUMMARY, which is the default option, or \*DETAIL. For most cases, \*SUMMARY provides all of the necessary analysis data.
- ▶ **FRCRCD:** This parameter allows the user to specify how often to force monitor records to the results file. For most cases, the default of \*CALC is acceptable. The user can specify a larger number to reduce the overhead of the DB Monitor; a smaller number increases the overhead.
- ▶ **COMMENT:** This parameter allows the user to provide a description of the collection. If the DB Monitor is started using the Start Performance Monitor (STRPFRMON) command, JOB(\*ALL) is used for the JOB option and data is placed in the QAPMDBMON file in the QPFRDATA library using the same member name as specified for the STRPFRMON command. Note that the user needs to use the End Database Monitor command (ENDDDBMON \*ALL) to end the DB Monitor for all jobs.

### End Database Monitor (ENDDDBMON) parameters

The End Database Monitor (ENDDDBMON) parameters and their functions are listed here:

- ▶ **JOB:** The user can specify a particular job name or end all jobs (\*ALL). If a particular job name is used, DB Monitor only ends the monitor that was started with that same job name. It is possible to end one monitor on a job and still have another monitor collecting on that same job.
- ▶ **COMMENT:** This parameter allows the user to provide a description of the data collection.

## Identifying and tuning problem areas

There are many different methods to identify problems and tune troublesome database statements. One of the most common methods is to identify the most dominating, time-consuming queries and work on each of them individually. Another method is to leverage global information and to use this information to look for indexes that are "begging" to be created.

It is usually best to first concentrate on repetitious non-reusable ODPs, table scans, and long index builds. Also, look for repetitious short-running queries that are not optimized well. Joins and sorts can be more difficult to analyze. If joins and sorts account for a significant portion of run time, they need to be addressed as well. Fine tuning smaller problems should be done after large problems have been addressed. Generally, indexes are used to tune most performance problem areas.

### 3.7.5 SQL Visual Explain

SQL Visual Explain is a tool used to visualize the optimization method selected by the database engine when a query is performed. Visual Explain provides a graphical representation and makes it easier for the analyst to determine actions to be taken (such as creating a new index for example).

Visual Explain can be used:

- ▶ On SQL statements stored in database performance monitor data
- ▶ On detailed monitor data (STRDBMON or Detailed SQL Performance Monitor)
- ▶ Via "Recent SQL Performance Monitors" task in Operations Navigator's SQL Script Center
- ▶ On SQL statements in the SQL Script Center

Monitor data has to be collected on V4R5 (can be another system) or later to guarantee all of the information is available for drawing the picture.

#### Running Visual Explain

Running Visual Explain from Run SQL Scripts is sometimes called *running Visual Explain proactively*, because you can run the query, make changes, and then run it again to see how the changes affected the implementation. However, if your query is long running, you may want to collect the data using a detailed SQL performance monitor and analyze it later using Visual Explain. You can explain a statement without running it, but the data gathered is based on estimates made by the Query Optimizer.

#### Running Visual Explain reactively

Running Visual Explain using detailed performance monitor data is sometimes referred to as *running Visual Explain reactively*, because the query explained has already run. However, if your query is long running, this may be your best option. In addition, the data collected by your detailed performance monitor can be more accurate than explaining a query from Run SQL scripts without actually running it.

1. In the Operations Navigator window, expand **your server->Database->SQL Performance Monitors**.
2. Right-click the detailed SQL performance monitor you want to use and select **List Explainable Statements**.
3. In the SQL statements monitored list, select the SQL statement you want to visually explain.
4. Click **Visual Explain** and the SQL statement is displayed, as a graph, in the Visual Explain window.

Figure 3-32 shows the example output from Visual Explain.

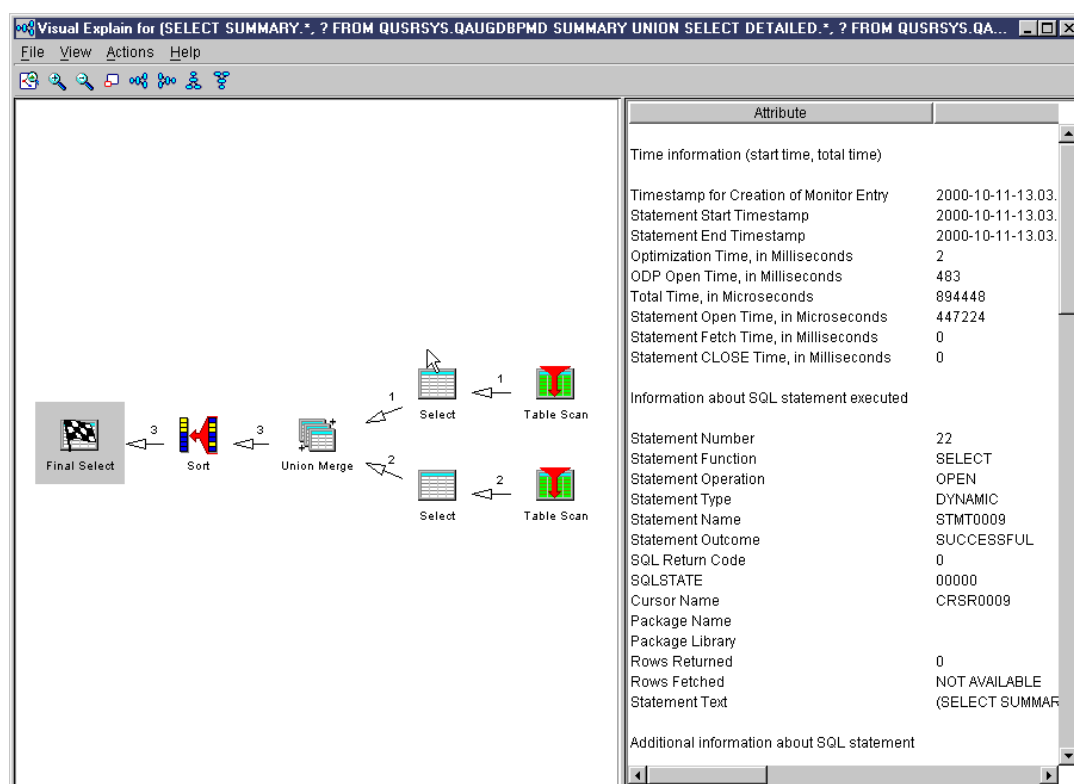


Figure 3-32 Example of a Visual Explain

## Creating an SQL performance monitor

Creating a new SQL performance monitor creates a new instance of a monitor on your system. You can have multiple instances of monitors running on you system at one time. However, there can only be one monitor instance monitoring all jobs. When collecting information for all jobs, the monitor will collect on previously started jobs or new jobs started after the monitor is created. However, when you end a performance monitor, the instance of the monitor is terminated and cannot be continued, where a paused monitor can be restarted. To create an SQL performance monitor, follow these steps:

1. In the Operations Navigator window, expand **your server->Database**.
2. Right-click **SQL Performance Monitor** and select **New**.
3. Select **Summary** or **Detailed**.
4. Specify the name you want to give the monitor in the Name field.
5. Specify the library in which you want to save the information that the monitor gathers in the Library field.
6. If you want to specify the maximum amount of system memory that the monitor is allowed to consume, specify the size in MB in the Storage (MB) field.
7. Click the **Monitored Jobs** tab.
8. If you want to monitor all of the available jobs and another monitor is not monitoring all jobs, select **All**. Only one monitor can monitor all jobs at a time.
9. If you want to monitor only certain jobs, select a job that you want to monitor in the Available jobs list and click **Select**. Repeat this step for each job that you want to monitor. Individual jobs can be monitored by only one active monitor at a time.

10. If you selected a job that you do not want to monitor or if a job you selected is already being monitored, select that job in the Selected jobs list and click **Remove**.
11. If you selected Summary monitor, click the **Data to Collect** tab.
12. On the Data to Collect tab, select the types of data that you want to collect. If you want to collect all types of data, click **Select All**.
13. Click **OK**. The performance monitor starts and runs until it is ended or paused.

## Executing Visual Explain from Run SQL Scripts

To run Visual Explain from Run SQL Scripts, follow these steps:

1. In the Operations Navigator window, expand **your server->Database**.
2. Right-click **Database** and select **Run SQL Scripts**.
3. If you have an existing SQL statement that you want to use, copy that statement and paste it into the Run SQL Scripts window.
4. If you do not have an existing SQL statement, type a statement in the Run SQL Scripts window.
5. Highlight the text of your statement and select **Explain** (or **Run and Explain**) from the Visual Explain menu. Or you can use the corresponding toolbar buttons.

If you choose *Explain* from the Visual Explain menu, the system only displays your query diagram without actually running it. If you choose *Run and Explain* from the Visual Explain menu, your query is run by the system before it is displayed. The results of the query (if any) are displayed in the Run SQL Scripts window. Running and displaying your query could take a significant amount of time, but the information displayed will be more complete and accurate.

For more information on Visual Explain, see *Advanced Functions and Administration on DB2 Universal Database for iSeries*, SG24-4249.





## Tuning iSeries for a WebSphere or Java environment

This chapter describes the performance and tuning aspects of OS/400. It looks at the tuning process for the OS/400 environment and the particular re-configuration that may be required for efficient WebSphere Application Server and Java execution. The specific details of tuning the WebSphere Application Server itself and of building Java applications that will perform most effectively follow in later chapters.

This chapter considers:

- ▶ OS/400 system values and settings that are required for a WebSphere/Java application
- ▶ The OS/400 resource access environment: This includes the OS/400 system jobs that you find executing to support the environment

For the OS/400 environment, we suggest or propose using values and settings that will assist in obtaining the best performance of the overall execution.

## 4.1 iSeries performance behavior

The performance of a server application is a combination of program instruction execution and access to system services. If you have a client-server application, you also have to take into consideration the network. Access to system services typically involves accessing databases. Traditionally, the AS/400 system has been an excellent performer within the commercial environment. Typical RPG, COBOL, or C commercial applications have performance profiles where fewer resources are spent processing high level language program instructions than system services such as database processing.

A commercial Java application is expected to have a similar profile, but currently, the program instructions portion is proportionately higher. This implies that more time is spent executing program instructions compared to performing system activities such as database I/Os.

In addition, Java makes extensive use of threads, which presents a challenge to the traditional OS/400 tuning practices.

## 4.2 Verifying the state of Licensed Program Products

Before you even consider looking at operating system parameters, you should ensure that the operating system and licensed programs involved in your application are up to date. Errors found in licensed programs are corrected with program temporary fixes (PTFs) that are released either as cumulative packages (CUM PTF), group PTFs, or on an individual basis. Performance fixes for Java are released frequently and you should ensure that you have them applied on your system. The best way to do so is to apply the latest CUM PTF package for your version of OS/400. You should also apply the group PTFs for:

- ▶ Database
- ▶ IBM Toolbox for Java
- ▶ WebSphere Application Server
- ▶ Java
- ▶ HTTP server

The group PTFs frequently contain PTFs, which are not yet released in the cumulative PTF package.

Use the Display Data Area (DSPDTAARA) command to determine the group PTF level on your system as shown in Figure 4-1.



```
Display Data Area (DSPDTAARA)

Type choices, press Enter.

Data area . . . . . _____ Name, *LDA, *GDA, *PDA
Library . . . . . *LIBL Name, *LIBL, *CURLIB
Output . . . . . * *, *PRINT
Output format . . . . . *CHAR *CHAR, *HEX

Additional Parameters

System . . . . . *LCL *LCL, *RMT

Bottom
F3=Exit F4=Prompt F5=Refresh F12=Cancel F13=How to use this display
F24=More keys
```

Figure 4-1 Entering the DSPDTAARA command

An example of the output created with the DSPDTAARA command is shown in Figure 4-2.

```
Display Data Area                                     System: AS05

Data area . . . . . : SF99069
Library . . . . . : QJAVA
Type . . . . . : *CHAR
Length . . . . . : 50
Text . . . . . :

Value
Offset *...+...1....+...2....+...3....+...4....+...5
0      'Group PTF#: SF99069-04 V5R1M0 August 15, 2001 '

Press Enter to continue.

F3=Exit F12=Cancel

Bottom
```

Figure 4-2 An example of DSPDTAARA command output

The number following the Group PTF identifier states the PTF level.

**Note:** These data areas only exist if a group PTF is installed.

The data areas to be verified are shown in Table 4-1 and Table 4-2.

Table 4-1 Data areas for various program products

Program product	OS/400 V4R5	OS/400 V5R1
Java	QJAVA/SF99068	QJAVA/SF99069
HTTP Server	QHTTPSVR/SF99068	QHTTPSVR/SF99156
Database	QSYS/SF99105	QSYS/SF99501

There are group PTFs available for WebSphere Application Server. The data areas displaying their level are described in Table 4-2.

Table 4-2 Data areas for WebSphere versions

WebSphere Application Server version	OS/400 V4R5	OS/400 V5R1
Advanced Edition 3.5	QEJB/SF99138	QEJB/SF99147
Advanced Edition 3.0	QEJB/SF99135 QEJB/SF99136	Not supported
Standard Edition 3.5	QEJB/SF99142	QEJB/SF99146
Standard Edition 3.0	QEJB/SF99133 QEJB/SF99134	Not supported
Advanced Edition 4.0	QEJBADV4/SF99239	QEJBADV4/SF99241
Advanced Edition 4.0 Single Server	QEJBADV4/SF99240	QEJBADV4/SF99242

**Note:** WebSphere Application Server group PTF includes a group PTF for Java, HTTP Server, and database.

“Referenced Web sites” on page 213 provides you with Web sites that can help you get the latest information about updates. You may also contact your software service provider to check these levels.

## 4.3 OS/400 system tuning

This section covers some of the basic operating system parameters, which influence the way your Java application performs on an iSeries server. Both system values and subsystem settings may have significant effects on the application performance. This section also covers general work management issues, such as setting the run priorities of your Java-based jobs and their associated threads.

### 4.3.1 Manual tuning versus automatic tuning

First, you have to decide whether you tune your system manually or automatically. You may decide to use the automatic tuning to provide you with a starting point for tuning the system. Do this by using the automatic adjustment during the heavy workload on your system and then turning it off. After you turn off the automatic adjuster, you may decide to adjust the pool sizes and activity level settings manually with 10% changes as shown in “Tuning main storage with the WRKSYSSTS command” on page 72.

In both cases, you must check the settings of system values manually because the only things the automatic tuner changes are the pool sizes and activity levels. This is determined with the setting of one system value – QPFRADJ. The setting of this system value provides you with the following levels of IBM provided automatic tuning:

- ▶ 0 = No adjustment
- ▶ 1 = Adjustment at IPL
- ▶ 2 = Adjustment at IPL and automatic adjustment
- ▶ 3 = Automatic adjustment

The difference between settings 2 and 3 is that when using the value of three, the tuner remembers the values that were used before the IPL and starts the IPL with these values. When using the value of 2, the system resets the memory pool sizes and activity levels and starts tuning with the default values. We recommend that you use the value 3 if you decide to use the automated tuning.

When you use manual tuning, you decide how much system resources are allocated to each subsystem and how many jobs are allowed to use the CPU at any given time.

### 4.3.2 Verifying the settings of system values

System values are to the jobs running on an iSeries server what properties are to Java. They define the basic characteristics of the operating system and all the jobs being supported by the operating system.

There are a number of system values you should consider changing with either the Change System Value (CHGSYSVAL) or Work with System Value (WRKSYSVAL) commands. The following sections describe each of these system values and the recommended settings. The recommended numbers have been found to work during hundreds of real life cases.

#### Allocation system values

Every job on the iSeries server has a Work Control Block Table (WCBT) entry that the operating system uses to keep track of the jobs in the system. A WCBT entry can be seen as a pre-fabricated frame for a job that is filled with the jobs' run attributes by the time the job becomes active within a subsystem. A job holds the WCBT entry as long as the job is active and as long as there exists even one spooled output file for the job on the system.

The allocation system values are:

- ▶ **QACTJOB**: Initial number of active jobs
- ▶ **QADLACTJ**: Additional number of active jobs
- ▶ **QADLSPLA**: Spooling control block additional storage
- ▶ **QADLTOTJ**: Additional number of total jobs
- ▶ **QJOBMSGQFL**: Job message queue full action
- ▶ **QJOBMSGQMX**: Maximum size of job message queue
- ▶ **QJOBMSGQSZ**: Job message queue initial size
- ▶ **QJOBMSGQTL**: Job message queue maximum initial size
- ▶ **QJOBSPLA**: Spooling control block initial size
- ▶ **QMAXJOB**: Maximum number of jobs
- ▶ **QMAXSPLF**: Maximum spooled files
- ▶ **QRCLSPLSTG**: Reclaim spool storage
- ▶ **QTOTJOB**: Initial total number of jobs

You should first verify the settings of the allocation system values on your system. In some cases, this might be the only action needed to remove the performance problem. The following sections discuss the most important system values and their settings.

### QTOTJOB (Initial total number of jobs)

This specifies the initial number of Work Control Block Table entries created during IPL.

You have two options to find out the required setting of this system value:

- ▶ WRKSYSSTS command
- ▶ DSPJOBTL command

Enter the WRKSYSSTS command and press F21 to set the assistance level to Advanced. A screen similar to the example in Figure 4-3 is displayed. Pay attention to the value shown in the Jobs in system field. Add 20% to the number and set it to be the new system value.

We recommend that you use the advanced assistance level because it provides all the required information at a glance.

Work with System Status									
								AS05	
								10/12/01	10:12:35
% CPU used . . . . .	:	97.8		System ASP . . . . .	:	87.74	G		
% DB capability . . . . .	:	12.5		% system ASP used . . . . .	:	28.9425			
Elapsed time . . . . .	:	00:00:03		Total aux stg . . . . .	:	144.7	G		
Jobs in system . . . . .	:	443		Current unprotect used . . . . .	:	3135	M		
% perm addresses . . . . .	:	.009		Maximum unprotect . . . . .	:	3756	M		
% temp addresses . . . . .	:	.015							
Sys	Pool	Reserved	Max	----DB----	--Non-DB---	Act-	Wait-	Act-	
Pool	Size M	Size M	Act	Fault Pages	Fault Pages	Wait	Inel	Inel	
1	493.68	200.95	+++++	.0 .0	.0 .0	.0	.0	.0	
2	2356.60	3.35	20	.0 .0	.0 .0	2990	.0	.0	
3	40.95	.00	5	.0 .0	.0 .0	.0	.0	.0	
4	504.75	.00	9	.0 .0	1.9 1.9	39.0	.0	.0	
5	200.00	.00	150	.0 .9	.0 .0	.0	.0	.0	
6	500.00	.99	200	.0 .0	.9 25.0	13153	.0	.0	
								Bottom	
====>									
F21=Select assistance level									

Figure 4-3 WRKSYSSTS with advanced assistance level setting

Enter the DSPJOBTL command and pay attention to the number shown in the “Entries in use” column. Add 20% to that value and set that to be your system value.

Figure 4-4 and Figure 4-5 show you what to expect when you enter the Display Job Tables (DSPJOBTL) command.

```

Display Job Tables
AS05
10/12/01 10:07:24

Permanent job structures:
Initial . . . . . 30
Additional . . . . 30
Available . . . . 441
Total . . . . . 864
Maximum . . . . . 163520

Temporary job structures:
Initial . . . . . 20
Additional . . . . 10
Available . . . . 26

-----Entries-----
Table      Size      Total  Available  In-use  Other
1          885504    864      441      443      0

Bottom

Press Enter to continue.

F3=Exit  F5=Refresh  F11=In-use entries  F12=Cancel

```

Figure 4-4 DSPJOBTL showing the total number of WCBT entries

In these examples, the value 30 represents the setting of the system value QTOTJOB and the value 20 represents the setting of the system value QACTJOB. The number of In-Use entries shows how many jobs are active and how many jobs are not active but have output files left on the system. Compare Figure 4-4 with Figure 4-5.

```

Display Job Tables
AS05
10/12/01 10:07:24

Permanent job structures:
Initial . . . . . 30
Additional . . . . 30
Available . . . . 441
Total . . . . . 864
Maximum . . . . . 163520

Temporary job structures:
Initial . . . . . 20
Additional . . . . 10
Available . . . . 26

-----In-use Entries-----
Table      Active      Job      Output
            Queue      Queue
1          271         0        172

Bottom

Press Enter to continue.

F3=Exit  F5=Refresh  F11=Total entries  F12=Cancel

```

Figure 4-5 DSPJOBTL showing the WCBT entries currently in use

The setting of this system value should be high enough not to be exceeded between IPLs. You should suspect an incorrect setting of this system value if your system is suffering from mysterious slowdowns happening periodically. If the number of jobs in the system exceeds the number specified with this system value, all the jobs in the system are paged out from the main storage and the amount of WCBT entries specified with system value QADLTOTJ is created.

After this is completed, all the jobs are paged back in to the main storage and the normal processing continues until the amount of the additional WCBT entries created is used up and the jobs are paged out again. Based on our experiences, we prefer to have the QTOTJOB set too large than too small. The default value shipped with the operating system is 30, which usually is too small. A change to this system value takes effect at the next IPL.

### **QADLTOTJ (Additional number of total jobs)**

This value specifies how many WCBT entries will be created if the amount of jobs in system exceeds the number specified with system value QTOTJOB. The default value of 10 should be enough provided that the setting of QTOTJOB is sufficient. The larger this value is, the longer it takes to create the additional WCBT entries. The normal workload on the system is not being processed while the additional entries are created. We suggest that you use the default setting to minimize the performance impact. A change to this system value takes effect immediately.

### **QACTJOB (Initial number of active jobs)**

This value specifies the initial number of active jobs for which storage is allocated during IPL. An active job is a job that has started running but has not ended. The amount of storage allocated for each active job is approximately 110K. This storage is in addition to the storage allocated using the system value QTOTJOB. A change to this system value takes effect at the next IPL. The shipped value is 20. A reasonable value to assign to QACTJOB is your estimate of the number of active jobs on a typically heavy-use day. This can be done by viewing the active jobs field on the active jobs display (WRKACTJOB command).

Work with Active Jobs						AS05
						10/12/01 10:13:42
CPU %:	99.1	Elapsed time:	00:05:43	Active jobs:	<b>271</b>	
Type options, press Enter.						
2=Change 3=Hold 4=End 5=Work with 6=Release 7=Display message						
8=Work with spooled files 13=Disconnect ...						
Opt	Subsystem/Job	User	Type	CPU %	Function	Status
	QBATCH	QSYS	SBS	.0		DEQW
	QCMN	QSYS	SBS	.0		DEQW
	QCTL	QSYS	SBS	.0		DEQW
	QSYSSCD	QPGMR	BCH	.0	PGM-QEZSCNEP	EVTW
	QEJBSBS	QSYS	SBS	.0		DEQW
	QEJBADMIN	QEJB	BCI	.0	PGM-QEJBADMIN	JVAW
	QEJBMNTR	QEJB	ASJ	.0	PGM-QEJBMNTR	EVTW
	TEAM16	QEJB	BCI	81.4	PGM-QEJBSVR	JVAW
	TEAM16ADMN	QEJB	BCI	.1	PGM-QEJBADMIN	JVAW
						More...
Parameters or command						
====>						
F3=Exit F5=Refresh F7=Find F10=Restart statistics						
F11=Display elapsed data F12=Cancel F23=More options F24=More keys						

Figure 4-6 Using WRKACTJOB to determine the QACTJOB system value setting

Set the QACTJOB system value high enough so that storage does not need to be allocated for additional active jobs using the system value QADLACTJ.

### ***QADLACTJ (Additional number of active jobs)***

This specifies the additional number of active jobs that will have storage allocated when the initial number of active jobs (the system value QACTJOB) is reached. An active job is a job that has started running but has not ended. Auxiliary storage is allocated whenever the number of active jobs exceeds the storage that has already been allocated. The amount of storage allocated for each job is approximately 110K. Setting the number close to 1 can cause frequent interruptions when many additional jobs are needed. The number should not be set too high because the time required to add additional storage should be minimized. A change to this system value takes effect immediately.

## **Storage system values**

This is a set of system values that affect to how the system storage is being used. The ones that you should set correctly are:

- ▶ QBASACTLVL
- ▶ QBASPOOL
- ▶ QMAXACTLVL
- ▶ QMCHPOOL

### ***QBASACTLVL (Base storage pool activity level)***

Increasing this value reduces or eliminates thread transitions into the ineligible state. The initial choice of value should be (arbitrarily) high, and then as implementation proceeds, monitor, and decrease the value if necessary.

This value determines the maximum number of threads (not jobs) running in the \*BASE pool that can use the processor concurrently. If the activity level is too low, the threads may transition to the ineligible condition. If the activity level is too high, excessive page faulting may occur.

It is important to increase this value for systems that are executing a large number of threads. Having a setting that is too low may slow down or even hang the system.

Threads running on the system can be in any of the following states:

- ▶ Active
- ▶ Wait
- ▶ Ineligible

An *active thread* exists in main storage and processes work requested by the application. A thread in the *wait state* needs a resource that is not available. An *ineligible thread* has work to do, but the system is unable to accept more work at that time because there is not an activity level available.

Use the WRKSYSSTS system command to adjust this setting. Increasing the value highlighted on the example shown in Figure 4-7 reduces or eliminates the thread transitions into the ineligible state.

The setting of QBASACTLVL only affects the activity level setting for the \*BASE pool. Use the WRKSHRPOOL or WRKSYSSTS commands to change the activity level settings for other pools.

Work with System Status								AS05	
								10/12/01	10:12:35
% CPU used . . . . . :	97.8	System ASP . . . . . :	87.74	G					
% DB capability . . . . . :	12.5	% system ASP used . . . . . :	28.9425						
Elapsed time . . . . . :	00:00:03	Total aux stg . . . . . :	144.7	G					
Jobs in system . . . . . :	443	Current unprotect used . . . . . :	3135	M					
% perm addresses . . . . . :	.009	Maximum unprotect . . . . . :	3756	M					
% temp addresses . . . . . :	.015								
Sys	Pool	Reserved	Max	----DB----	--Non-DB--	Act-	Wait-	Act-	
Pool	Size M	Size M	Act	Fault	Pages	Fault	Pages	Wait	Inel
1	493.68	200.95	+++++	.0	.0	.0	.0	.0	.0
2	2356.60	3.35	20	.0	.0	.0	.0	2990	1329
3	40.95	.00	5	.0	.0	.0	.0	.0	.0
4	504.75	.00	9	.0	.0	1.9	1.9	39.0	.0
5	200.00	.00	150	.0	.9	.0	.0	.0	.0
6	800.00	.99	200	.0	.0	.9	25.0	13153	.0

Figure 4-7 Using WRKSYSSTS to change the QBASACTLVL

**Note:** By default, everything, except interactive jobs and printer jobs, runs in the \*BASE pool. This includes the WebSphere Application Server jobs.

If you are using a system with a mixed load of traditional HLL code and Java code, you should consider creating a separate pool for the Java work. An example of doing this is shown in 4.3.3, “Creating a separate memory pool for a subsystem” on page 67.

### **QBASPOOL (Base storage pool minimum size)**

This specifies the minimum value of main storage that is left to the \*BASE pool after all the subsystems are started. If the setting of this value is too high there will be not enough storage left for the additional memory pools. Having a too low amount of memory in \*BASE pool may have a negative effect on your subsystem monitors, communications jobs, and so on.

### **QMAXACTLVL (Maximum activity level of system)**

This specifies the number of threads that can compete at the same time for main storage and processor resources. For all active subsystems, the sum of all threads running in all memory pools cannot exceed QMAXACTLVL. If a thread cannot be processed because the activity level has been reached, the thread is held until another thread reaches a time slice end or a long wait. In Java or WebSphere environments, this usually generates a transaction rollback. This value should be larger than the sum of the activity levels for all your memory pools. If QMAXACTLVL is smaller, activity levels in the memory pools may not be used. We recommend the setting \*NOMAX.

A change to this system value takes effect immediately.



### **QMCHPOOL (Machine storage pool size)**

This is perhaps the most important system value because this value specifies the amount of main storage that is reserved for both the operating system programs and Licensed Internal Code. Enter the WRKSYSSTS command as shown in Figure 4-3 on page 62. Look at the “Reserved Size” column for memory pool 1 (\*MACHINE), multiply that number by 2, and set that number to this system value.

Your system performance may be severely affected if this system value is too small. Your application performance may be unsatisfactory if this value is too large. There may be enough main storage available in the system to support your workload but is not where it is needed.

**Note:** The system values QTOTJOB and QACTJOB require an IPL for the changes to become effective.

### **4.3.3 Creating a separate memory pool for a subsystem**

iSeries main memory (main storage) is analogous to Random Access Memory (RAM) on personal computers. On the iSeries server, all main storage can be divided into logical allocations called *memory pools* (storage pools). A pool is a division of main or auxiliary storage (memory). The two types of memory pools in a system are:

- ▶ **Shared pools:** A shared memory pool is a pool in which multiple subsystems can run jobs. There are 63 memory pools that you may define system wide, but you may only define up to ten memory pools for one subsystem.

Shared pools are either special or general. The machine pool and base pool are considered special shared pools; all other shared pools are considered general.

- ▶ **Private pools:** A private memory pool is a pool in which a single subsystem can run jobs. Private pools are pools of main storage that cannot be shared by multiple subsystems. A private pool, often simply referred to as a *pool*, contains a specified amount of storage to be used by only one subsystem.

Four memory pools are defined by default:

- ▶ \*MACHINE for iSeries system jobs
- ▶ \*BASE: Contains the minimum value specified by the QBASPOOL system value plus any unassigned main memory. By default, WebSphere Application Server jobs run in this storage pool.
- ▶ \*INTERACT for interactive jobs
- ▶ \*SPOOL for print jobs

The reason for creating a separate memory pool for a subsystem is to make sure that the jobs running in a subsystem are provided with system resources when they need them. If the job requests an object that is not in the main memory, the object has to be retrieved from a disk. This is called a *page fault*, since the system always brings in whole memory pages (4 KB of size) rather than individual objects. A page fault adds 10 to 30 milliseconds to end user response time, based on the time to read data from a disk into main memory.

As an example, we create a separate memory pool for the QEJBSBS, the subsystem on which our WebSphere Application server is running. We use a shared memory pool to enable the application to take advantage of the Expert Cache function provided by IBM where the system automatically determines the best approach for handling data in the memory pool. See “Enabling the Expert Cache” on page 73 for details on this subject.

## Directing the QEJSBS jobs to a pool of their own

The steps to direct jobs in a subsystem to a memory pool of their own are outlined here:

1. Create a memory pool to be used for the subsystem jobs.
2. Enter the following command to tell the subsystem that it has a new memory pool available:

```
CHGSBSD SBSD(QEJB/QEJSBS) POOLS((2 *SHRPOOLX))
```

3. Enter the following command to direct the jobs to the new memory pool:

```
CHGRTGE SBSD(QEJB/QEJSBS) SEQNBR(9999) POOLID(2)
```

4. Restart the subsystem.

The above steps are shown in detail here:

1. Enter the WRKSHRPOOL command to select a pool and set the initial size and activity level.

The Work with Shared Pools screen appears as shown in Figure 4-8.

Work with Shared Pools						
				System: AS05		
Main storage size (M) . :				4096.00		
Type changes (if allowed), press Enter.						
Pool	Defined Size (M)	Max Active	Allocated Size (M)	Pool ID	-Paging Option-- Defined Current	
*MACHINE	493.68	+++++	493.68	1	*FIXED	*FIXED
*BASE	2056.60	40	2056.60	2	*CALC	*FIXED
*INTERACT	504.75	9	504.75	4	*CALC	*FIXED
*SPOOL	40.95	5	40.95	3	*FIXED	*FIXED
*SHRPOOL1	200.00	150	200.00	5	*CALC	*FIXED
<b>*SHRPOOL2</b>	<b>800.00</b>	<b>200</b>		<b>6</b>	<b>*CALC</b>	*FIXED
*SHRPOOL3	.00	0			*FIXED	
*SHRPOOL4	.00	0			*FIXED	
*SHRPOOL5	.00	0			*FIXED	
*SHRPOOL6	.00	0			*FIXED	
More...						
Command						
==>						
F3=Exit F4=Prompt F5=Refresh F9=Retrieve F11=Display tuning data						
F12=Cancel						

Figure 4-8 Creating a separate memory pool for QEJSBS

In this example, we select the \*SHRPOOL2 and assign 800 MB of main storage to the pool. We also define the initial activity level of 200. These are just initial values, but we refine them later by using the WRKSYSSTS command.

2. Use the CHGSBSD command to specify the pool identifier to be used by the subsystem.

By default, subsystem pool 1 is where the subsystem monitor will run. We suggest using the \*BASE pool for this pool identifier and using pool numbers from two to ten for the actual jobs running on the subsystem. Separating the subsystem monitor from the jobs may save you from performing an IPL in case one of the jobs encounters a problem from which it cannot recover. For example, it may be impossible to provide resources for the subsystem monitor job if a looping application program and subsystem monitor are competing for the memory in the same pool.

Figure 4-9 shows you an example of using the Change Subsystem Description command.

```

Change Subsystem Description (CHGSBSD)

Type choices, press Enter.

Subsystem description . . . . . SBSD          > QEJBSBS
Library . . . . .                      > QEJB
Storage pools:                                POOLS
  Pool identifier . . . . .                > 2
  Storage size . . . . .                  > *SHRPOOL2
  Activity level . . . . .
Maximum jobs . . . . . MAXJOBS             *SAME
Text 'description' . . . . . TEXT          *SAME

Sign-on display file . . . . . SGNDSPF      *SAME
Library . . . . .
Subsystem library . . . . . SYSLIBLE        *SAME

                                                                 Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys

```

Figure 4-9 CHGSBSD QEJBSBS display

3. Enter the CHGRTGE command as shown in Figure 4-10 to direct the jobs to the pool created in the previous step.

```

Change Routing Entry (CHGRTGE)

Type choices, press Enter.

Subsystem description . . . . . QEJBSBS      Name
Library . . . . . QEJB                     Name, *LIBL, *CURLIB
Routing entry sequence number . 9999         1-9999
Comparison data:
  Compare value . . . . . *SAME

  Starting position . . . . .                1-80, *SAME
Program to call . . . . . *SAME              Name, *SAME, *RTGDTA
Library . . . . .                          Name, *LIBL, *CURLIB
Class . . . . . *SAME                       Name, *SAME, *SBSD
Library . . . . .                          Name, *LIBL, *CURLIB
Maximum active routing steps . . *SAME       0-1000, *SAME, *NOMAX
Storage pool identifier . . . . 2            1-10, *SAME

                                                                 Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys

```

Figure 4-10 CHGRTGE to direct the jobs to the pool

With the CHGRTGE command, we specify that all the jobs in the QEJBSBS that use the routing entry with sequence number 9999 will go to the subsystems memory pool 2.

4. Restart the subsystem, by using ENDSBS command, followed by STRSBS command.

The CHGRTGE command does not affect the jobs already active within a subsystem, it only affects the jobs that start after the routing entry is started. To direct all the active jobs to the newly created memory pool, you must either end the subsystem and restart it or stop the jobs and restart them.

### 4.3.4 Setting the memory pool sizes and activity levels manually

The manual adjustment can be done with using both the WRKSHRPOOL and WRKSYSSTS commands. This section shows you how to use the WRKSYSSTS command to tune the system. All the screen captures were done with using the assistance level of *advanced*.

#### Determining the memory pool size

If data is already in main storage, it can be referred to independently of the memory pool in which it is located. However, if the necessary data does not exist in any memory pool, it is brought into the same memory pool for the job that referred to it (this is known as a *page fault*).

As data is transferred into a memory pool, other data is displaced and, if changed, it is automatically recorded on disk (this is called paging). The memory pool size should be large enough to keep data transfers (into and out of main storage) at a reasonable level as the rate affects performance.

1. Determine in which pool the QEJBSBS jobs are running.
  - a. Enter the WRKSBS command to see the screen shown in Figure 4-11.

```

Work with Subsystems
System:  AS05

Type options, press Enter.
  4=End subsystem    5=Display subsystem description
  8=Work with subsystem jobs


```

Opt	Subsystem	Total Storage (M)	-----Subsystem Pools-----									
			1	2	3	4	5	6	7	8	9	10
	QBATCH	.00	2									
	QCMN	.00	2									
	QCTL	.00	2									
	QEJBSBS	.00	2	<b>6</b>								
	QHTTPSVR	.00	2									
	QINTER	.00	2	4								
	QSERVER	.00	2									
	QSPL	.00	2	3								
	QSYSWRK	.00	2	5								
	QUSRWRK	.00	2									

```

Bottom

Parameters or command
===>
F3=Exit   F5=Refresh   F11=Display system data   F12=Cancel
F14=Work with system status

```

Figure 4-11 Using the WRKSBS command to relate the memory pools with WRKSYSSTS

- b. Press F14 key or enter the WRKSYSSTS command for the screen displayed in Figure 4-12. Note that pool 2 for subsystem QEJBSBS is *not* pool 2 on the WRKSYSSTS display. The memory pools are allocated in the order the subsystems are started. In this case, pool 6 on the WRKSYSSTS display is pool 2 for the QEJBSBS.

Work with System Status								AS05			
								10/12/01	10:12:35		
% CPU used . . . . . :				97.8	System ASP . . . . . :				87.74 G		
% DB capability . . . . . :				12.5	% system ASP used . . . . . :				28.9425		
Elapsed time . . . . . :				00:00:03	Total aux stg . . . . . :				144.7 G		
Jobs in system . . . . . :				443	Current unprotect used . . . . . :				3135 M		
% perm addresses . . . . . :				.009	Maximum unprotect . . . . . :				3756 M		
% temp addresses . . . . . :				.015							
Sys	Pool	Reserved	Max	---DB----		--Non-DB--		Act-	Wait-	Act-	
Pool	Size M	Size M	Act	Fault Pages		Fault Pages		Wait	Inel	Inel	
1	493.68	200.95	+++++	.0	.0	.0	.0	.0	.0	.0	
2	2356.60	3.35	20	.0	.0	.0	.0	2990	.0	.0	
3	40.95	.00	5	.0	.0	.0	.0	.0	.0	.0	
4	504.75	.00	9	.0	.0	1.9	1.9	39.0	.0	.0	
5	200.00	.00	150	.0	.9	19.0	40.0	.0	.0	.0	
6	800.00	.99	200	.0	.0	10.9	125.0	13153	.0	.0	

5. The rate of change is important here. Increase activity level if the rate of transitions from *wait-to-ineligible* is approaching the rate of *active-to-wait*.
6. Increase or decrease the activity levels with ten percent increments.
7. Remember that Java and WebSphere threads need more activity levels per pool than the legacy applications used to need.

Work with System Status								AS05		
								10/12/01	10:12:35	
% CPU used . . . . . :	97.8	System ASP . . . . . :						87.74	G	
% DB capability . . . . . :	12.5	% system ASP used . . . . . :						28.9425		
Elapsed time . . . . . :	00:00:03	Total aux stg . . . . . :						144.7	G	
Jobs in system . . . . . :	443	Current unprotect used . . . . . :						3135	M	
% perm addresses . . . . . :	.009	Maximum unprotect . . . . . :						3756	M	
% temp addresses . . . . . :	.015									
Sys	Pool	Reserved	Max	----DB----	--Non-DB--		Act-	Wait-	Act-	
Pool	Size M	Size M	Act	Fault	Pages	Fault	Pages	Wait	Inel	Inel
1	493.68	200.95	+++++	.0	.0	.0	.0	.0	.0	.0
2	2356.60	3.35	20	.0	.0	.0	.0	2990	.0	.0
3	40.95	.00	5	.0	.0	.0	.0	.0	.0	.0
4	504.75	.00	9	.0	.0	1.9	1.9	39.0	.0	.0
5	200.00	.00	150	.0	.9	19.0	40.0	.0	.0	.0
6	800.00	.99	200	.0	.0	10.9	125.0	13153	.0	.0
								Bottom		
====>										
F21=Select assistance level										

Figure 4-13 Using WRKSYSSTS to monitor the transition data

See *OS/400 Work Management*, SC41-3306, for a thorough description of system values and other work management related functions. Here we provide you with the minimum information needed to get your system up and running.

## Tuning main storage with the WRKSYSSTS command

The Work with System Status (WRKSYSSTS) command provides a list of page faulting and wait-to-ineligible (W->I) transitions for each main memory pool. Set the assistance level to intermediate or advanced. Use the following steps to determine acceptable levels of faulting and W->I transitions:

1. The machine pool (pool 1) should have fewer than 10 faults per second (sum of DB and non DB faults). Increase the pool by 10% until the faults per second are satisfactory.
2. If your machine pool is experiencing very low page fault rates (<0.4), you should decrease the size of the machine pool. If the page fault rate is this low, this may affect work in some other pools.
3. If only system jobs and subsystem programs are running \*BASE, the fault rate for that pool should be less than 30 faults per second. You need to decrease another pool to increase \*BASE.
4. For interactive pools, the W->I should be small (less than 10% of A->W). If you see any W->I, increase the MAXACT by 5-10 until the W->I is 0. After you increase the MAXACT value, press F10 to reset the statistics; do not use F5 to refresh. You should wait at least one minute between refreshes.

5. For user pools, the fault rate alone is not necessarily a measure of good or poor performance. Response time and throughput are the actual measures of performance. Therefore, you should tune your pools by moving storage from pools with better performance to pools with poor performance. Continue to move the storage to the pool with poor performance until performance improves. Do not decrease a pool by more than 10% at one time.

### ***Enabling the Expert Cache***

Expert Cache is a function that is shipped with the operating system. Using it may improve your systems performance noticeably, especially if your applications access data sequentially.

When using Expert Cache, the operating system dynamically determines which objects or portions of objects should remain in a shared main storage pool based on the reference patterns of data within the object. It also determines which objects should have larger blocks of data brought into main storage. The decision is based on how frequently the object is accessed. If the object is no longer accessed heavily, the system automatically makes the storage available for other objects that are accessed. If the newly accessed objects become heavily accessed, the objects have larger blocks of data placed in main storage. The system automatically determines the best approach for handling data in the memory pool based on the following criteria:

- ▶ If many threads are running in a small memory pool, the system limits the amount of memory used by each thread.
- ▶ If the memory pool for those threads has enough memory, the system determines (object by object) how much data to bring into the memory pool.
- ▶ If objects are referred to sequentially, the system brings larger blocks of data into memory and delays writing changes of the data. This reduces the number of I/O operations issued by the job and reduces the contention for disk drives, which, in turn, reduces the time that jobs wait on I/O requests.
- ▶ If objects are referred to randomly, the system does not bring in large blocks of data because that does not reduce the number of I/O operations.

Enabling Expert Cache is easy:

1. Enter the WRKSYSSTS command.
2. Press F11 for paging data.
3. Replace \*FIXED with \*CALC for the required pool or pools.

## **4.3.5 Common WebSphere Application Server and Java jobs**

This section identifies some of the common WebSphere- and Java-related jobs and their corresponding subsystems. You can use the information in Table 4-3 to help determine where your workload issues may be.

*Table 4-3 Jobs and subsystems related to Java and WebSphere*

Subsystem	Job name	Job type	Description
The QEJBSBS subsystem holds the WebSphere Application Server <i>Version 3</i> related jobs. Generally all WebSphere instances will run in this subsystem.	<instance name>	BCI	Application server running job. All WebSphere Java code, Servlets, JSPs and EJBs execute in this job.
	<instance name>ADMN	BCI	Administration server for the instance.
	<instance name>MNTR	BCH	Monitor (nanny) job.

Subsystem	Job name	Job type	Description
The QEJBADV4 subsystem holds the WebSphere Application Server <i>Version 4.0 Advanced Edition</i> related jobs. Generally all WebSphere instances will run in this subsystem.	<instance name>	BCI	Application server running job. All WebSphere Java code, Servlets, JSPs and EJBs execute in this job.
	<instance name>ADMN	BCI	Administration server for the instance.
	<instance name>MNTR	BCH	Monitor (nanny) job.
The QEJBAES4 subsystem holds the WebSphere Application Server <i>Version 4.0 Single Server Edition</i> related jobs. Generally all WebSphere instances will run in this subsystem.	<instance name>	BCI	Holds all WebSphere Application Server processes including an instance, administration server, and monitor.
The QHTTSPVR subsystem holds the HTTP server related jobs. Generally all HTTP server instances will run in this subsystem.	<instance name>	BCH	Primary instance manager for the HTTP server.
	<instance name>	BCI	Usually several of these jobs, each capable of managing incoming requests.
The QINTER subsystem holds the interactive jobs. Since a traditional interactive job is unable to execute threads, running a Java program from an interactive prompt will result in a Batch Immediate job being spawned.	QJVACMSRV	BCI	Java invoked by Interactive Job.
The QSYSWRK subsystem holds system related jobs. Jobs in this subsystem are generally pre-start jobs.	QSQSRVR	PJ	Usually a large number of these jobs, which provide the database pooling for SQL access to DB2.
	QRWTSRVR	PJ	Used for DDM access to Database files.
The QSERVER subsystem holds jobs related to communication.	QZRCRSVS	PJ	Used for RPC (also known as DPC).
The QSERVER subsystem holds jobs related to the TCP/IP servers.	QZDASOINIT	PJ	Provides database pooling for the Toolbox JDBC driver.

### 4.3.6 Working with the prestart jobs

A prestart job is a batch job that starts running before a program on a remote system sends a program start request. Prestart jobs are different from all the other jobs because they use prestart job entries to determine which program, class, and memory pool to use when they are started. Within a prestart job entry, you specify attributes that the subsystem uses to create and manage a pool of prestart jobs.



The prestart jobs QSQSRVR and QZDASOINIT are the ones that perform JDBC access to system resources such as database that your Java applications request. The native JDBC functions are carried out by the QSQSRVR jobs running on QSYSWRK, where the IBM Toolbox uses the QZDASOINIT jobs that are running on QSERVER subsystem. Make sure that these server jobs have the resources they need when they need them or the users will experience bad response times or even time outs. The easiest way to provide these server jobs with the memory and activity level is to direct these jobs to a memory pool that no other jobs are using. We use the QSQSRVR jobs here as an example, but the same approach may be used with the QZDASOINIT jobs.

In this section, we improve the server jobs performance by answering the following list of questions:

- ▶ How many prestart jobs do you have running on the system?
- ▶ How many prestart jobs are started with the subsystem?
- ▶ How many prestart jobs do you need to process the workload?
- ▶ How do you provide resources for the prestart jobs?

Based on the characteristics and the ongoing work to improve the prestart jobs, we suggest that you verify (and refresh if necessary) the level of the database group PTF every three months. Do this by pointing your browser to: <http://www.as400service.ibm.com/>

Select **Fixes-> Group PTFs-> OS/400 level** for information about the group PTF package available. If your system is on back level, order the current package through the channels that you normally use.

## Determining the number of active prestart jobs

Before you change the number of prestart server jobs, you should know how many of these jobs you need. By default, these server jobs run in the subsystem QSYSWRK. The easiest way to decide this is by entering the Display Active Prestart Jobs (DSPACTPJ) command as shown in the Figure 4-14.

Display Active Prestart Jobs			AS05
			10/12/01 09:04:14
Subsystem . . . . .	QSYSWRK	Reset date . . . . .	10/10/01
Program . . . . .	QSQSRVR	Reset time . . . . .	11:34:03
Library . . . . .	QSYS	Elapsed time . . . . .	0045:30:11
Prestart jobs:			
Current number . . . . .			38
Average number . . . . .			38.1
Peak number . . . . .			52
Prestart jobs in use:			
Current number . . . . .			34
Average number . . . . .			34.0
Peak number . . . . .			49
Program start requests:			
Current number waiting . . . . .			0
Average number waiting . . . . .			.0
Peak number waiting . . . . .			0
Average wait time . . . . .			00:00:00.0
Number accepted . . . . .			4619
Number rejected . . . . .			0

Figure 4-14 Display Active Prestart Jobs

When comparing these values with the values shown in Figure 4-17, you can decide that the default values are not sufficient enough.

### Determining the number of prestart jobs you start

This is shown in the subsystem descriptions prestart job entries part. You enter the Display Subsystem Description command as shown here:

```
DSPSBSD SBSD(QSYSWRK)
```

Then you see a screen like the example in Figure 4-15.

Display Subsystem Description

System: AS05

Subsystem description: QSYSWRK      Library: QSYS

Status: ACTIVE

Select one of the following:

1. Operational attributes

2. Pool definitions

3. Autostart job entries

4. Work station name entries

5. Work station type entries

6. Job queue entries

7. Routing entries

8. Communications entries

9. Remote location name entries

10. **Prestart job entries**

More...

Selection or command

==>10

F3=Exit    F4=Prompt    F9=Retrieve    F12=Cancel

Figure 4-15 Display Subsystem Description

Choose option **10** (Prestart job entries) as shown in Figure 4-16.

Display Prestart Job Entries			System: AS05
Subsystem description: QSYSWRK		Status: ACTIVE	
Type options, press Enter.			
5=Display details			
Opt	Program	Library	User Profile
	QANEAGNT	QSYS	QUSER
	QIWVPPJT	QIWS	QUSER
	QRWTSRVR	QSYS	QUSER
<b>5</b>	<b>QSQSRVR</b>	<b>QSYS</b>	<b>QUSER</b>
	QSRRATBL	QSYS	QUSER
	QTMSRVR	QTCP	QTCP
	QTMSCLCP	QTCP	QTCP
	QTSSRCP	QTCP	QTCP
	Q5BWHSRV	QSYS	QUSER
			Bottom
F3=Exit F9=Display all detailed descriptions F12=Cancel			

Figure 4-16 Display Prestart Job Entries

Choose option 5 (Display details) for the prestart job entry QSQSRVR as shown on Figure 4-17.

Display Prestart Job Entry Detail		System: AS05
Subsystem description: QSYSWRK		Status: ACTIVE
Program . . . . .	:	QSQSRVR
Library . . . . .	:	QSYS
User profile . . . . .	:	QUSER
Job . . . . .	:	QSQSRVR
Job description . . . . .	:	*USRPRF
Library . . . . .	:	
Start jobs . . . . .	:	*NO
Initial number of jobs . . . . .	:	<b>5</b>
Threshold . . . . .	:	2
Additional number of jobs . . . . .	:	2
Maximum number of jobs . . . . .	:	*NOMAX
Maximum number of uses . . . . .	:	200
Wait for job . . . . .	:	*YES
Pool identifier . . . . .	:	2
<b>Class</b> . . . . .	:	QSYSCLS20
Library . . . . .	:	QSYS
Number of jobs to use class . . . . .	:	*CALC
Class . . . . .	:	*NONE
Library . . . . .	:	
Number of jobs to use class . . . . .	:	*CALC
Press Enter to continue.		
F3=Exit F12=Cancel F14=Display previous entry		

Figure 4-17 Display Prestart Job Entry Detail

On this screen, you see that the operating system starts five QSQSRVR prestart jobs when subsystem QSYSWRK is being started. When comparing this display with Figure 4-14 on page 75, you see that 38 additional prestart jobs were started during the time that the subsystem was active.

We recommended that you start the required amount of prestart jobs during the time of starting the subsystem (usually during IPL) because the overhead by that time is barely noticeable. Usually it is a good practice to start more prestart jobs than what you actually need. This is because the overhead of having too many prestart jobs is much less than the overhead of creating the prestart jobs on the fly.

### **Determining the amount of prestart jobs you need**

The amount of prestart jobs needed is based on the application needs. If you have an application that uses a large number of database connects and disconnects, you will need a lot of prestart jobs available. If your application uses a relatively small number of database connects and disconnects, you will need a relatively small amount of these prestart jobs on your system. The same applies to the number of concurrent users; if there is only a handful of concurrent users, you need fewer prestart jobs than you need when you have hundreds of users on the application at the same time. Usually, the default value shipped with the operating system is not sufficient.

Enter the DSPACTPJ command and examine the output as shown in Figure 4-14 on page 75 to find out the number of prestart jobs you have currently running. Make sure that you start at least that amount of jobs when your subsystem is being started.

### **Specifying the number of prestart jobs started**

This is done by using the CHGPJE command, as shown in Figure 4-18.

Change Prestart Job Entry (CHGPJE)		
Type choices, press Enter.		
Subsystem description . . . . .	SBSD	> QSYSWRK
Library . . . . .		*LIBL
Program . . . . .	PGM	> QSQSRVR
Library . . . . .		*LIBL
User profile . . . . .	USER	*SAME
Start jobs . . . . .	STRJOBS	*SAME
Initial number of jobs . . . . .	INLJOBS	> XXXXX
Threshold . . . . .	THRESHOLD	> XXXXX
Additional number of jobs . . . .	ADLJOBS	YYYYY
Maximum number of jobs . . . . .	MAXJOBS	*SAME
Job name . . . . .	JOB	*SAME
Job description . . . . .	JOB	*SAME
Library . . . . .		
Maximum number of uses . . . . .	MAXUSE	*SAME
Wait for job . . . . .	WAIT	*SAME
Pool identifier . . . . .	POOLID	> 2
Class:	CLS	
Class . . . . .		> YOURCLASS
Library . . . . .		> YOURLIB
Number of jobs to use class . . .		*SAME
Class . . . . .		*SAME
Library . . . . .		
Number of jobs to use class . . .		*SAME
F3=Exit F4=Prompt F5=Refresh F12=Cancel F13=How to use this display		
F24=More keys		

Figure 4-18 Change Prestart Job Entry command

Note that you can specify the prestart jobs to use a class different from the one provided with the operating system. The default is to use the class QSYSCLS20. You may also direct the prestart jobs to use a memory pool that is different from the system specified default of \*BASE. See 4.3.3, “Creating a separate memory pool for a subsystem” on page 67, for details on doing this.

### Why use the private memory pool for QSQSRVR jobs

When you specify a pool with memory and activity level, and you direct the jobs to that pool, you make sure that no other job in the system can steal neither memory nor activity levels from a server job.

This is extremely important because the QSQSRVR jobs are the ones that actually perform the database-related part of work for the application as well as WebSphere Application Server administration server to access the information in the repository. If you reduce the number of activity levels available for the server jobs, you actually strangle the work that is done by the WebSphere Application Server.

## Providing a memory pool for prestart jobs

First, create a new memory pool in main storage as described in 4.3.3, “Creating a separate memory pool for a subsystem” on page 67. Use the WRKSYSSTS command to find out how much memory is available on the system and then decide where to remove it from. By default, all the memory resides in the \*BASE pool, except the amount specified for operating system, interactive workload and spooler.

## Directing the QSQSRVR jobs to a pool of their own

Perform the following steps to provide the server jobs with a private memory pool:

1. Enter the WRKSHRPOOL command and choose a pool number that is not used. See Figure 4-19 for an example of the WRKSHRPOOL display.
2. Specify the memory pool size, activity level setting, and paging option for SHRPOOLX.

Work with Shared Pools						
Main storage size (M) . . :				System: AS05		
4096.00						
Type changes (if allowed), press Enter.						
Pool	Defined Size (M)	Max Active	Allocated Size (M)	Pool ID	-Paging Defined	Option-- Current
*MACHINE	493.68	+++++	493.68	1	*FIXED	*FIXED
*BASE	2356.60	20	2356.60	2	*FIXED	*FIXED
*INTERACT	504.75	9	504.75	4	*FIXED	*FIXED
*SPOOL	40.95	5	40.95	3	*FIXED	*FIXED
*SHRPOOL1	200.00	150	200.00	5	*FIXED	*FIXED
*SHRPOOL2	.00	0			*FIXED	*FIXED
*SHRPOOL3	.00	0			*FIXED	
*SHRPOOL4	.00	0			*FIXED	
*SHRPOOL5	.00	0			*FIXED	
*SHRPOOL6	.00	0			*FIXED	
						More...
Command						
==>						
F3=Exit F4=Prompt F5=Refresh F9=Retrieve F11=Display tuning data						
F12=Cancel						

Figure 4-19 WRKSHRPOOL display

3. Use the following command to tell the subsystem that it has a new memory pool attached to it:
 

```
CHGSBSD SBSD(QSYSWRK) POOLS((2 *SHRPOOLX))
```
4. Use the following command to direct the server jobs to the newly attached memory pool:
 

```
CHGPJE SBSD(QSYSWRK) PGM(QSQSRVR) POOLID(2)
```
5. End and restart the QSYSWRK subsystem or end and restart the server jobs:
  - a. ENDPJ SBSD(QSYSWRK) PGM(QSQSRVR)
  - b. STRPJ SBSD (QSYSWRK) PGM(QSQSRVR)

## 4.4 Java system environment

If your iSeries server is only running Java, there is a limited amount of impact to gain from changing job priorities. However, in a mixed mode environment, the RUNPTY and Time slice settings become important. To determine the base settings, you need to understand how OS/400 determines the RUNPTY for each thread.

### 4.4.1 Java thread priority

The actual RUNPTY setting for a thread depends on the Java thread priority. In Java, you have the ability to set a thread priority ranging from 1 (lowest) through to 10 (highest). A standard thread runs at Java priority 5. OS/400 uses this setting to determine the runtime priority. The calculation is as follows:

Java Priorities: 1 (MIN\_PRIORITY) to 5 (NORM\_PRIORITY) to 10 (MAX\_PRIORITY)

OS/400 RUNPTY Priority = BCI RUNPTY + 11 - Java priority

(Interactive default BCI job priority is 14, Batch default BCI job priority is 44)

To change the RUNPTY setting for all the WebSphere jobs, use:

```
CHGCLS QEJB/QEJBCLS RUNPTY(...)
```

### 4.4.2 Time slice setting

Time slice is another opportunity to assign more processing time to your threads. Each thread has a separate CPU count, and each thread hits the Time Slice End (TSE) independently.

To change the setting for WebSphere, use:

```
CHGCLS QEJB/QEJBCLS TIMESLICE(...)
```


## 4.5 AnyNet

Systems that have AnyNet support enabled may see worse performance when running WebSphere applications than those that do not have AnyNet support enabled. By default, AnyNet support is not enabled. To check if AnyNet support is enabled on your system, use the DSPNETA command. To disable AnyNet support (if it is not used by other applications), use:

```
CHGNETA ALWANYNET(*NO)
```







## Tuning HTTP server and WebSphere Application Server

This chapter covers some of the more important tuning parameters of the HTTP server and WebSphere Application Server. Significant performance gains can be obtained simply by ensuring that settings are correctly configured for the expected load on the application. With each parameter, we outline some recommended values and indicate where to set them using the WebSphere Administrator's console.

This chapter covers the following topics:

- ▶ Tuning HTTP server (Web server)
- ▶ Adjusting WebSphere system queues
- ▶ Additional WebSphere tuning parameters
- ▶ Settings that should remain unchanged on the iSeries server

**Note:** This chapter is oriented to WebSphere Application Server *Version 3.5* with some mention of WebSphere Application Server Version 4 where necessary.

## 5.1 Tuning the HTTP server

Every client's request for a WebSphere Application Server-based application goes to the HTTP server first. Therefore, the HTTP server's performance plays a role in the overall system performance. This section examines some common HTTP server performance issues. For more information, refer to *AS/400 HTTP Server Performance and Capacity Planning*, SG24-5645.

### 5.1.1 Caching

The IBM HTTP Server for iSeries supports local caching since V4R3. Local caching can make a sizable difference in performance. With local caching, your Web server is basically pre-approved for "file open" tasks when the server is started. The actual objects may be on disk or in main storage. Certainly, a read from main memory is much faster than from disk. However, you have no guarantees where the data resides, unless you have an extremely large amount of memory on your server.

With caching, a hash table is built on the iSeries server listing all cached objects. For each HTTP object request, this table is consulted to determine if any of the requested objects are cached. If you compare caching to non-caching, cached pages take less CPU and should be retrieved substantially faster. Non-cached pages incur a slight performance hit because the cache hash table must be consulted.

Caching works best and is easiest to implement if it is applied to the most frequently accessed objects, such as the HTML pages and selected graphics files in a particular directory on the server. Also, once a cached object is changed, it is removed from the cache table. Objects that are subject to editing or changes are not good cache candidates.

V4R4 introduced a concept called *dynamic caching*. This can provide greater flexibility than static caching because the most frequently accessed objects can be cached, rather than having to statically specify the objects when you start the server. This tends to work best for small Web sites with a limited amount of objects that will be potentially served. However, on larger sites, the dynamic caching algorithm may use more server resources to manage the cache than what is saved by retrieving objects in the cache.

From a sizing perspective, caching can reduce a server load substantially in some cases. The key criteria is that certain HTML files and graphics files are accessed much more frequently than others and these should be cached. If most objects on your site are accessed with equal likelihood, caching will be of little benefit and can actually be detrimental. If you have certain heavily accessed pages, then caching should be helpful. However, you need to ensure that your server has adequate main memory (and low page fault rates). Also, your cached content must be relatively stable. You may want to be conservative in doing your sizing analyses.

To tune HTTP server caching, perform the following steps:

1. Open the HTTP server administration console from your Web browser. It usually runs on port 2001. Therefore, in your Web browser command line, you type:

```
http://systemname:2001
```

See Figure 5-1.



Figure 5-1 HTTP server console

2. On the left-hand side of the console, click **Configuration and Administration**.
3. On the left-hand side, click **Configuration** and then expand **System Management**.
4. Click **Local Caching** to see the page shown in Figure 5-2.

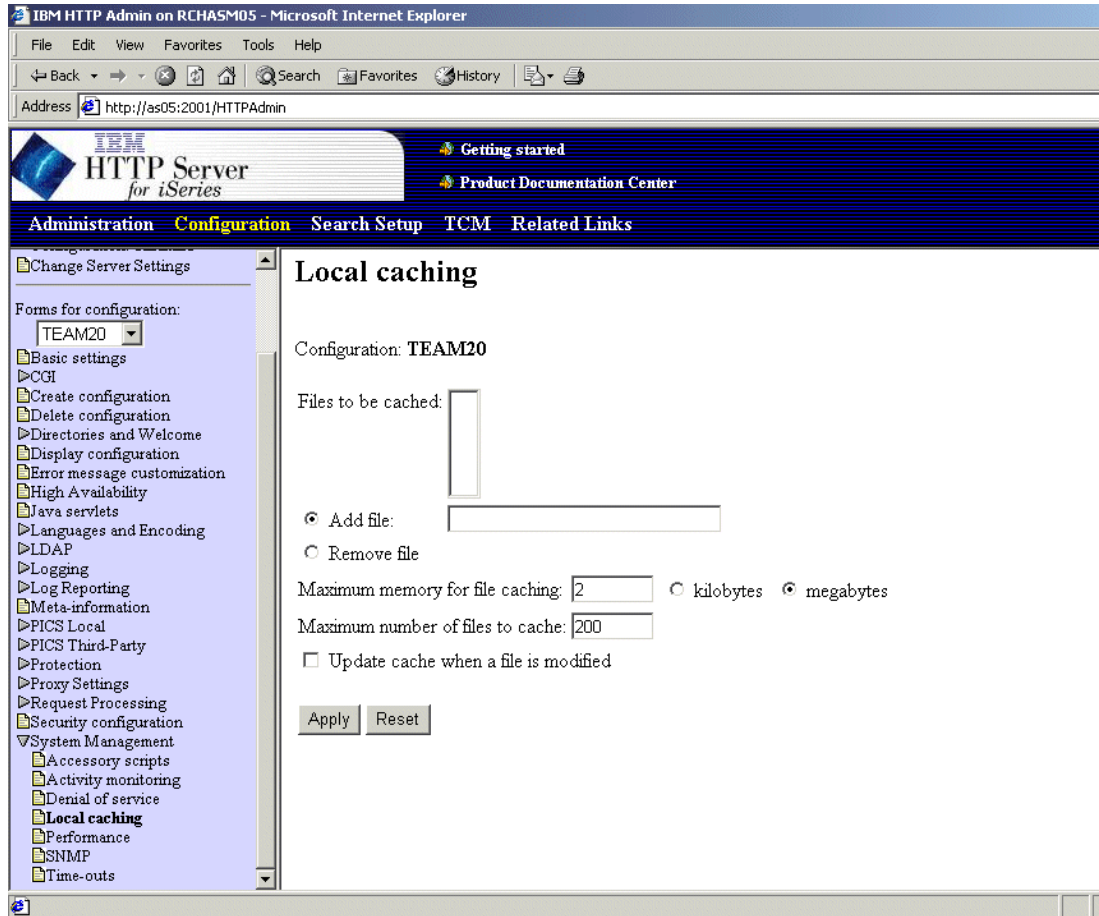


Figure 5-2 HTTP server local caching

5. Specify the file names, maximum number of files to be cached, and the cache size.

### 5.1.2 Persistent connections

HTTP server for iSeries supports persistent connections in V4R3 and later releases. However, be cautious in using these connections, because there are a number of scenarios that may negatively impact performance, for example:

- ▶ The browser may be slow in closing the request, which leaves the connection open and the server thread unavailable for other uses.
- ▶ The server keeps the request active until the persistence timeout is reached if the maximum requests per connection has not been realized.
- ▶ If persistence values are set too high, and all the server threads are occupied, new client requests will time-out, which has a major negative impact.

Persistent connections are most beneficial in a LAN intranet-oriented environment where response time is generally less than the persistence timeout. In fact, many commercial Internet sites turn off persistent HTTP connections completely. From an iSeries server perspective, you may want to set the maximum requests per connection parameter to 1 to avoid problems for Internet users.

To set the persistent connections, click **Performance** on the left-hand side of the page shown in Figure 5-2. Then you see the page shown in Figure 5-3.

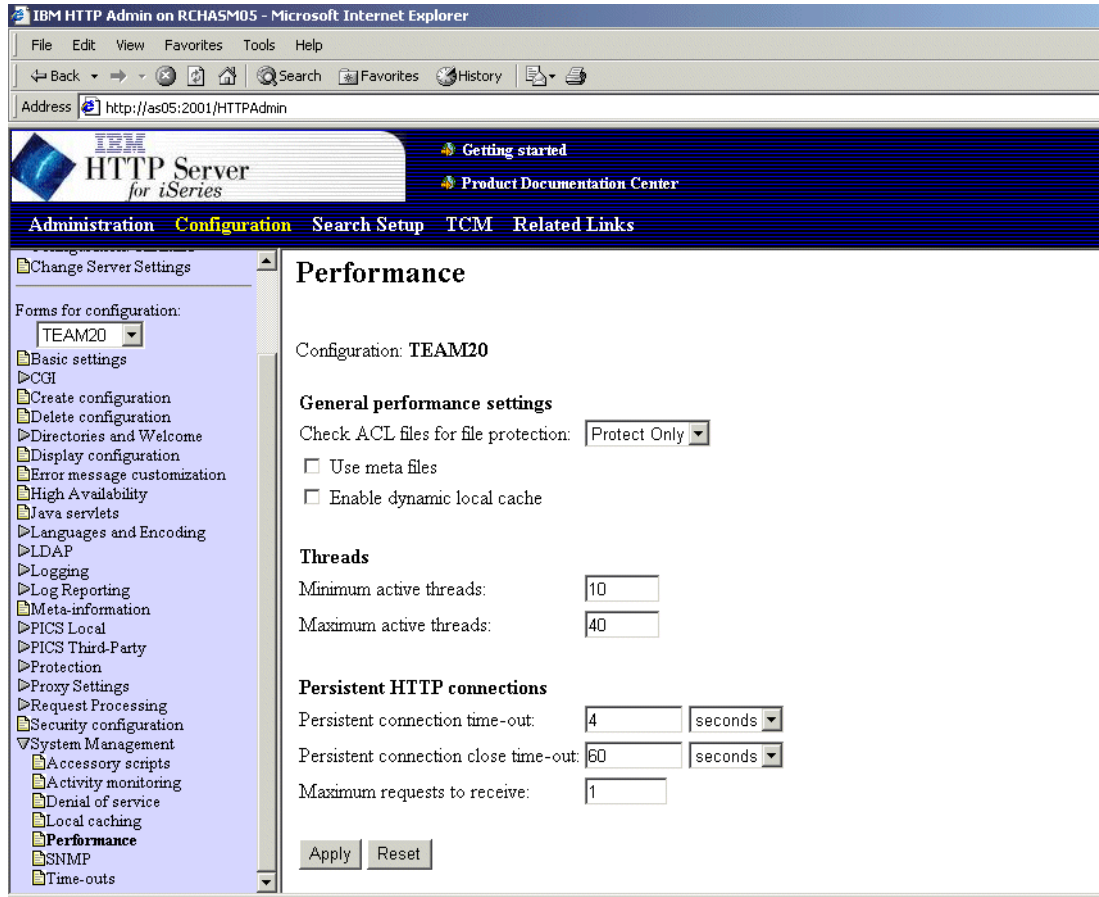


Figure 5-3 HTTP server persistent connections

Update the Maximum requests to receive field to 1.

### 5.1.3 Secure Sockets Layer (SSL)

If SSL client authentication is configured, the server requests the client's certificate for any HTTPS (HTTP + SSL) request. This of course adds overhead and, therefore, decreases system performance. If you don't need SSL, you can disallow SSL connections in your HTTP server console. Click **Security Configuration** on the display shown in Figure 5-3 and make sure that the **Allow SSL connections** check box is *not* selected (Figure 5-4).

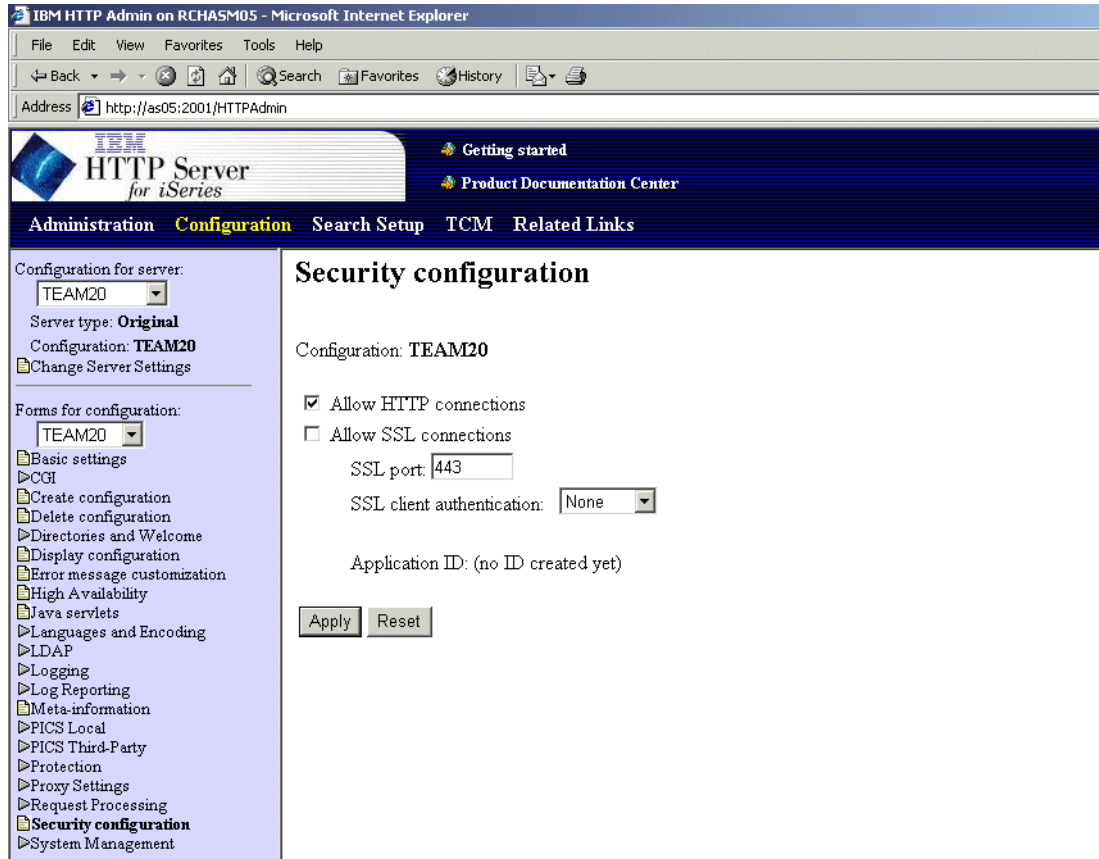


Figure 5-4 Deselecting the Allow SSL connections option

## 5.1.4 Maximum number of active HTTP server threads

Setting the maximum number of active threads in your HTTP server may affect the overall system performance. By default, this number is set to 40. When you change this number, you should change it in conjunction with setting the WebSphere Application Server servlets queue size and connection pool size.

### Setting the maximum number of active threads

Setting the HTTP server queue size is done via the HTTP server administration utility:

1. On the main screen, click **IBM HTTP Server for AS/400**.
2. Click **Configuration and Administration** (Figure 5-1 on page 85). Then you see the page shown in Figure 5-5.

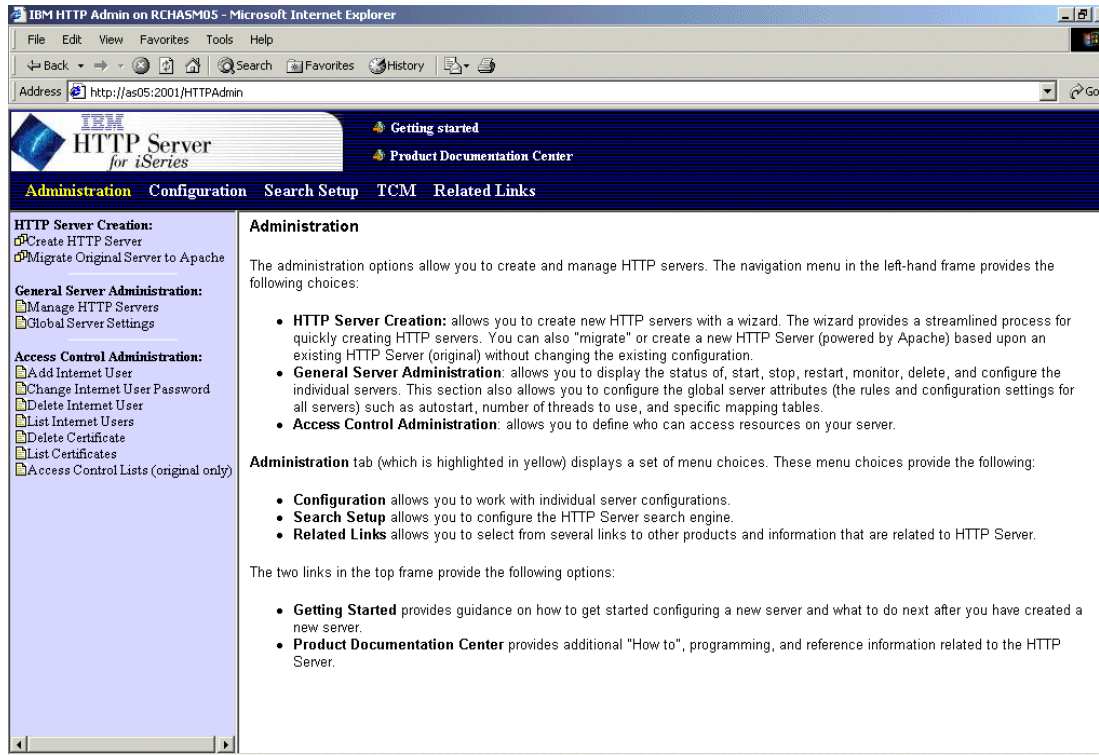


Figure 5-5 HTTP administration and configuration

3. Click **Configuration** at the top of the page.
4. Expand **System Management** on the left-hand side of the screen. Scroll down until you see **Performance** and click it (Figure 5-6).

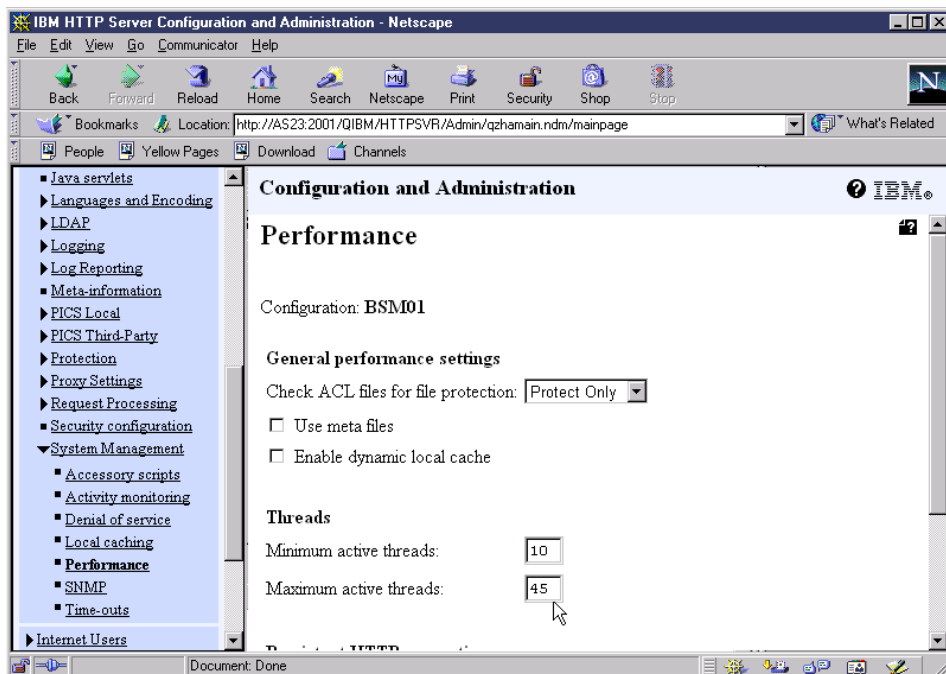


Figure 5-6 Setting the HTTP server threads

- Set the value in Maximum active threads field accordingly. Setting this number too high may cause WebSphere to be flooded with a mass of requests that cannot be satisfied. It is better for those requests to wait outside in the network. You can find guidelines on how to set this field in 5.3, “WebSphere queueing network” on page 93.

## Monitoring the number of active threads

The HTTP server allows you to monitor your current number of active threads and other activity statistics through the HTTP Server Monitor. You start the Monitor in the following way:

- On the page shown in Figure 5-5, click **Administration**. Then on left-hand side of the page, click **Manage HTTP Servers**. You should see the Manage HTTP Servers page as shown in Figure 5-7.

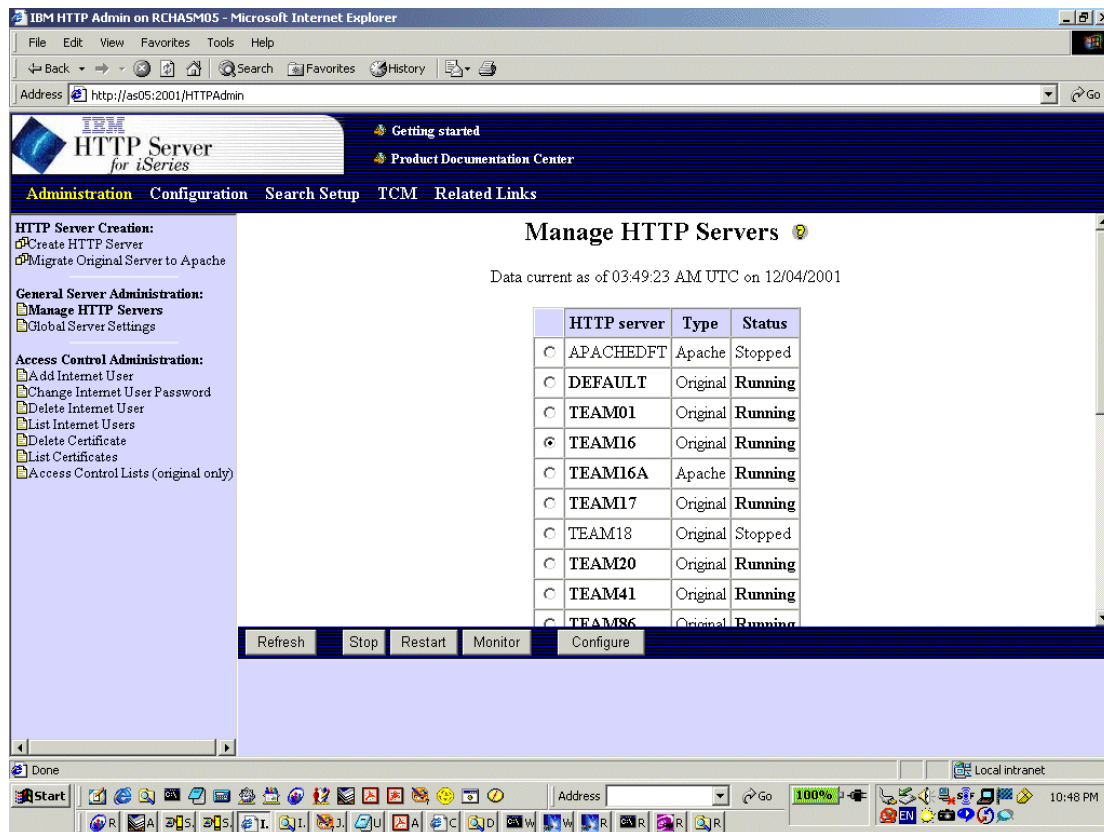


Figure 5-7 Manage HTTP Servers

- Select your HTTP server instance and click the **Monitor** button. You see a page similar to the one shown in Figure 5-8.



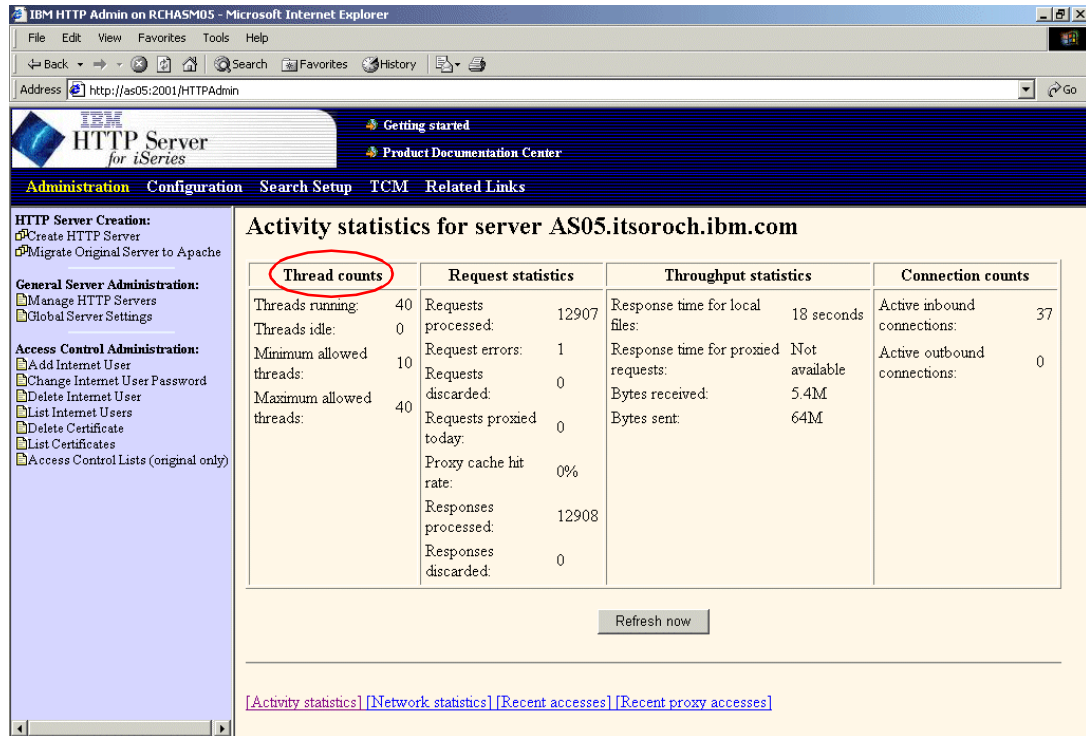


Figure 5-8 HTTP Server Monitor

The Thread counts column (circled) shows the current maximum and minimum active threads.

### 5.1.5 IBM HTTP Server (powered by Apache)

In OS/400 V5R1, IBM HTTP Server for iSeries (licensed product 5722-DG1), comes with two different HTTP servers:

- ▶ IBM HTTP Server (original)
- ▶ IBM HTTP Server (powered by Apache)

You can install both of them and run them at the same time. Powered by Apache should perform better, on average. While this does not necessarily mean that your WebSphere Application Server-based application will perform better, you may want to consider using powered by Apache for performance reasons. However, make sure first that IBM HTTP Server (powered by Apache) functionally supports all your business needs.

## 5.2 Direct execution (DE) versus JIT mode

You can run your WebSphere Application Server applications in JIT or DE mode. Which of these two will be used is controlled through Java Virtual Machine properties in the *admin.properties* file.

**Note:** In WebSphere Application Server Version 3.5.4, the DE mode with optimization level 40 typically produces slightly better performance than JIT. However, in Version 4, the performance of these two modes are comparable. This section shows you how to use DE instead of JIT mode in *Version 3.5.4*.

When you are ready to go to a production environment, use the Java Transformer. You should create the Java program objects using the CRTJVAPGM command. Place the Java program objects in the non-reloadable classpath. This means that the program objects are not reloadable. If you want to make a change to an object, you must stop and restart the server for it to be operational. The big advantage, from a performance point of view, is the program objects are persistent. That is, they continue to exist even if you end the server. This is not the case when you place objects in the reloadable classpath.

Because all WebSphere components (servlets, JSPs, and enterprise beans) are loaded by an *application classloader*, WebSphere components do not use the direct execution (DE) capabilities of the OS/400 JVM. Java program (JVAPGM) objects are not used when running WebSphere component code. Classes loaded by a *system classloader* or *WebSphere runtime classloader* use DE capabilities and JVAPGM objects.

By default, application servers run with the Java system property `java.compiler=jitc_de`. This property is set by the `ejbsvr.java.compiler` property in the `admin.properties` file. The *jtc\_de* value means that just-in-time compilation is done for any Java object for which a JVAPGM object does not exist (or cannot be used, in the case of WebSphere application classloaders). This setting provides the best overall performance. It is possible to change application servers to use direct execution instead of JIT mode. Doing so results in longer startup times, because creation of the JVAPGM takes longer than creation of the JIT stubs. After the JVAPGMs are created, performance of direct execution may be slightly improved over JIT.

To make an application server use direct execution for all classes, follow these steps:

1. Change the default value for all Application Server Instances so that individual application server settings are used.

In the `admin.properties` file, change the `ejbsvr.java.compiler=jitc_de` setting to `ejbsvr.java.compiler=NONE`.

2. Configure specific settings for each application server:

For application servers that will use direct execution, add the property `os400.defineClass.optLevel` with a value that indicates the optimization level to the System Properties of the JVM Settings on the application server. The `os400.defineClass.optLevel=xx` (where *xx* is the optimization level, for example 40) property needs to be provided on the command line arguments property for each application server, specified as `-Dos400.defineClass.optLevel=xx`. If you have an Application Server Instance for which you want to continue to use the JIT compiler, the `ejbsvr.java.compiler=jitc_de` property should be provided on the command line arguments property for that application server, specified as `-Djava.compiler=jitc_de`.

You can explicitly optimize your Java programs by using the Create Java Program (CRTJVAPGM) command from the OS/400 command line. This command runs the transformer against the specified Java bytecodes. However, using this command is only useful for Java objects loaded by the iSeries system classloader. For objects loaded through customized class loaders (for example, the WebSphere Web application class loader), the presence of the hidden program cannot be detected. Using CRTJVAPGM with customized class loaders provides no optimization.

For untransformed Java objects that are in the iSeries system classpath (and therefore loaded through the system class loader), the `os400.defineClass.optLevel` property can be specified to cause the byte codes to be implicitly transformed. The transformed version persists across multiple instantiations of the Java Virtual Machine. Note that if you specify this property, performance may be slowed by creation of the hidden program the first time you run a Java program. For Java objects that are loaded through customized class loaders (such as

the WebSphere Web application class loader), the `os400.defineClass.optLevel` Java Virtual Machine property will cause behavior similar to what the `os400.optimization` property does for Java objects in the system classpath. The main difference is that the transformed version is not saved. That is, each time the Java object is loaded, the transformation occurs again.

## 5.3 WebSphere queuing network

The WebSphere Application Server has a series of interrelated components that must be harmoniously tuned to support the custom needs of your end-to-end e-business application. These adjustments will help your system achieve maximum throughput, while maintaining overall system stability.

WebSphere Application Server establishes a *queuing network*, which is a network of interconnected queues that represent the various components of the application serving platform. These queues include the network, Web server, servlet engine, Enterprise JavaBean (EJB) component container, data source, and possibly a connection manager to a custom backend system. Each of these WebSphere resources represents a queue of requests waiting to use that resource.

The WebSphere queues are load-dependent resources. The average service time of a request depends on the number of concurrent clients and the average resource utilization.

### 5.3.1 Closed queues versus open queues

Most of the queues that make up the WebSphere queuing network are closed queues. A *closed queue* places a limit on the maximum number of requests active in the queue. (Conversely, an *open queue* places no such restrictions on the maximum number of requests active in a queue.) A closed queue allows system resources to be tightly managed. For example, the WebSphere servlet engine's Max Connections setting controls the size of the servlet engine queue. If the average servlet running in a servlet engine creates 10 MB of objects during each request, then setting Max Connections to 100 would limit the memory consumed by the servlet engine to approximately 1 GB. Therefore, closed queues typically allow system administrators to manage their applications more effectively and robustly.

In a closed queue, a request can be in one of two states – *active* or *waiting*. In the *active* state, a request is doing work or is waiting for a response from a downstream queue. For example, an active request in the Web server is either doing work (such as retrieving static HTML) or waiting for a request to complete in the servlet engine. In *waiting* state, the request is waiting to become active. The request remains in a waiting state until one of the active requests leaves the queue.

All Web servers supported by the WebSphere Application Server software are closed queues. The WebSphere servlet engine and data source are also closed queues in that they allow you to specify the maximum concurrency at the resource. The EJB container inherits its queue behavior from its built-in Java technology object request broker (ORB). Therefore, the EJB component container, like the Java ORB, is an open queue. Given this fact, it is important for the application calling enterprise beans to place limits on the number of concurrent callers into the EJB container.

Because the Servlet Redirector function, typically used as a means of separating the Web server and servlet engine on different machines is an EJB Client application, it is an open queue.

If enterprise beans are being called by servlets, the servlet engine will limit the number of total concurrent requests into an EJB container because the servlet engine has a limit itself. This fact is only true if you are calling enterprise beans from the servlet thread of execution. There is nothing to stop you from creating your own threads and bombarding the EJB container with requests. This is one of the reasons why it is not a good idea for servlets to create their own work threads.

### 5.3.2 Queue settings

As with any queuing system, you need to be aware of the following trade off:

*The more slots you make available in a queue, the better the service time is likely to be. On the other hand, the higher the number of slots there is, the larger amount of computing resources (memory, CPU) the queue is likely to use.*

That's why it is important to assign a fair value to the number of slots in WebSphere's queues. Values that are too low can result in a long service time (and therefore impact response time and throughput). Values that are too high may cause an unnecessary allocation of computing resources that won't therefore be available to perform other work (such as the processing in the Application Server). This situation can severely impact performance.

### 5.3.3 Tuning the queues

Figure 5-9 shows the overall structure of the WebSphere queuing mechanisms (queuing network). Three queues are involved in the process and as previously mentioned; they are all closed.

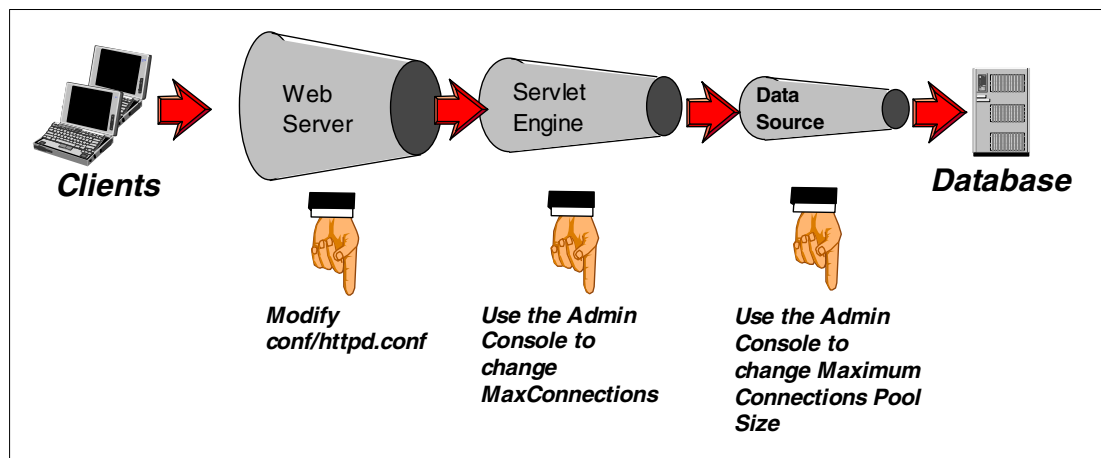


Figure 5-9 WebSphere queuing network

The Web Server queue controls how many concurrent HTTP requests can be served. If this queue is full, any further requests will be forced to wait in the network.

The servlet engine queue controls how many concurrent threads can be activated in the servlet. If the queue is flooded, further requests will be forced to wait in the HTTP server.

The data source queue controls how many concurrent database connections can be active at one single time. Overloading this queue causes servlets to wait for access.

It is a good practice to tune the queues from back to front – that is, assign a decreasing number of available slots as the transactions proceed down to the database. This funneling technique increases the chances that the requests that appear at a certain queue are ready to be processed and reduces the number of requests actually queuing up, at the same time optimizing the allocation of resources.

### Setting the HTTP server threads

The IBM HTTP Server's queues can be customized by setting an appropriate value for the maximum number of concurrent threads that you want to allow.

It is a good practice to allow for a number somewhat higher (but not significantly higher) than the maximum application concurrency – that is, the maximum number of clients that your environment can manage without experiencing throughput degradation or a prohibitive increase of the response time. The HTTP server may sometimes handle significant static page load, in addition to requests to servlets, and an allowance needs to be made for this traffic. A "maximum active threads" value that is too low may unnecessarily limit static page handling.

The maximum number of concurrent threads should typically be higher than the total number of connections a servlet is allowed to handle. For a further explanation, see the following section.

### Setting the servlet queue size

The maximum number of concurrent connections a servlet is allowed to handle should typically be lower than the total number of http threads, for a couple of reasons.

First of all, not all the requests take the same time to be fulfilled. Some requests may be very light and require a limited amount of computing power. Some HTTP requests, depending on the configuration, may end up simply requesting an HTML page, for example. Secondly, we want to funnel a smaller number of requests to the next queue, the data source queue, and as we mentioned, ideally we want to populate that queue with requests that are ready to be served.

The servlet queue size is set through WebSphere Administrator Console. In the console, follow these steps:

1. Expand your node.
2. Expand your application server.
3. Click **Default Servlet Engine**.
4. Click the **Advanced** tab.
5. Set the servlet queue size by specifying the appropriate number in the Max Connections field (Figure 5-10).

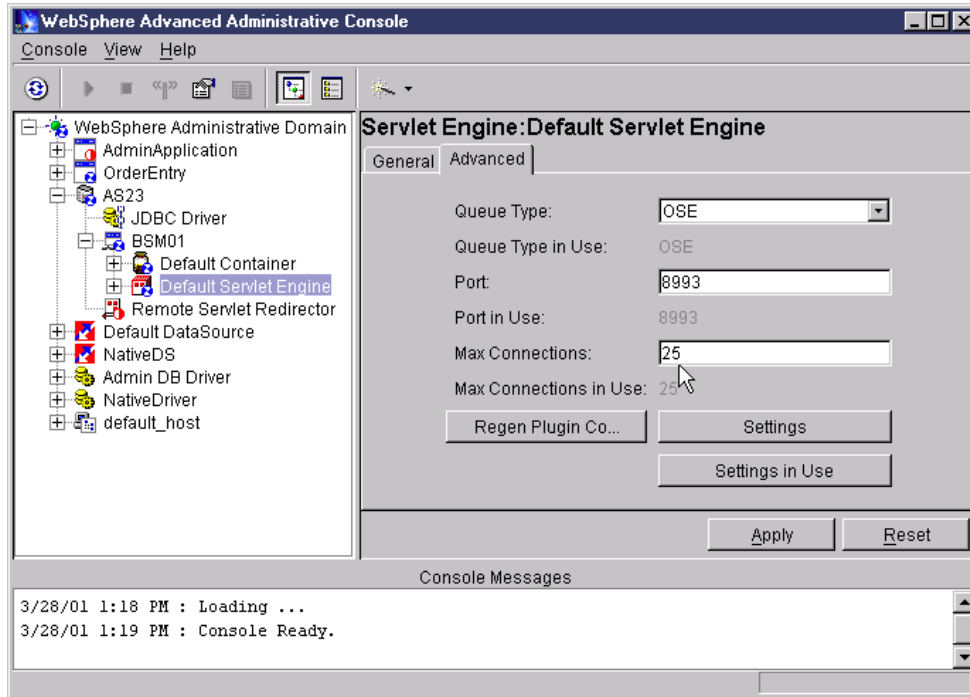


Figure 5-10 Setting the servlet queue size

## Setting the connection pool size

The maximum connection pool size that you want to choose depends on two factors:

- ▶ How many clients you expect to be connected and served at the same time
- ▶ How much memory and CPU you want to devote to the data queues

There is no easy way to tell what this number should be; measuring and testing is probably the best way to go about. Keep in mind the following considerations: a data source connection takes 2 to 4 MB of main storage. In a memory constrained system, this can quickly add up to a significant chunk of the whole available storage – hindering other work on the system. Also, managing the connections takes CPU cycles. Therefore, there is no need to exaggerate with the maximum number of connections. Remember that not all of the requests that get to the servlet engine will generate the same database workload; some requests may, in fact, not perform any DB I/O at all.

Using the WebSphere Resource Analyzer (see 3.6.1, “WebSphere Application Server Resource Analyzer” on page 38), you can measure how many connections were actually used and how long the competing requests had to wait before being granted a connection. Keep the number of connections to a minimum that is compatible with your response time objectives.

The connection pool size is set through the WebSphere Administrator's Console. In the console, follow these steps:

1. Click your data source.
2. Click the **Advanced** tab.
3. Set the connection pool size by specifying the appropriate number in the Maximum Connection Pool size field (Figure 5-11).

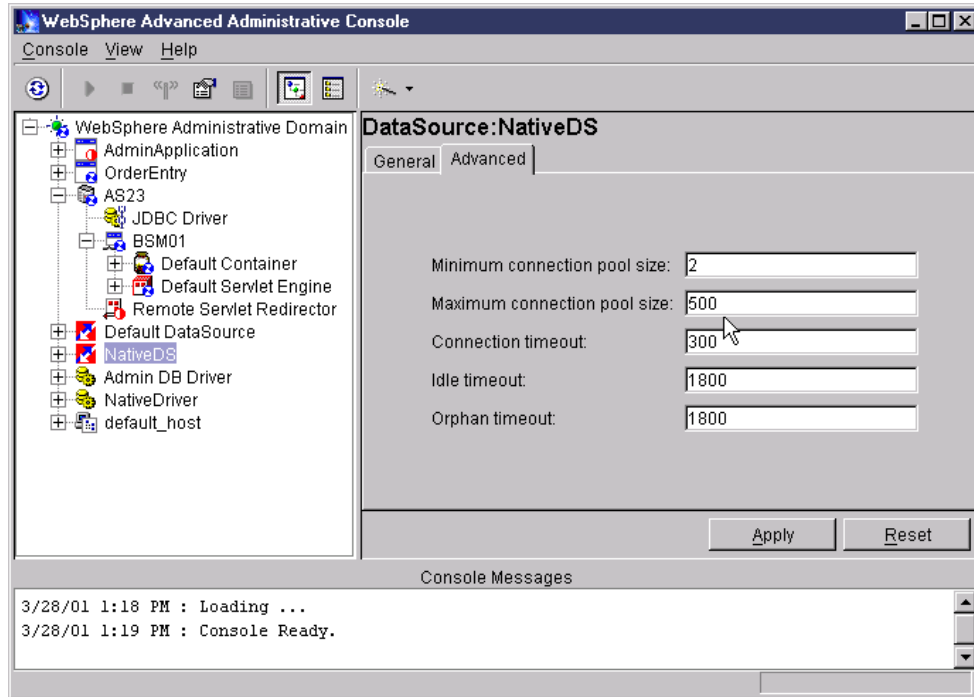


Figure 5-11 Setting the connection pool size

## 5.4 Garbage collection initial size

The best way to optimize garbage collection is through testing. A setting that is too large will cause the garbage collector to run less frequently, but it will need to collect more objects. About 10 to 15 seconds between garbage collections should produce a good result.

Setting the initial garbage collection size to 32 Mb for one- or two-way systems creates a reasonable *starting point for testing*. For 4- to 8-way systems, we recommend you *start* with 256 Mb, and for 12- to 24-way systems, *start* with 1 Gb. After that, experiment by increasing and decreasing this number to determine the optimum initial size for your particular environment.

You set the garbage collection initial size in the command line of WebSphere Administrative Console. In the console, follow these steps:

1. Expand your node.
2. Click your application server.
3. Click the **General** tab (Figure 5-12).
4. Set the value for parameter `-Xms` to specify the garbage collection initial size. The value `-Xms256m` in Figure 5-12 specifies the initial size of 256 MB. To specify the size to 64 MB, set this value to `-Xms64m`.

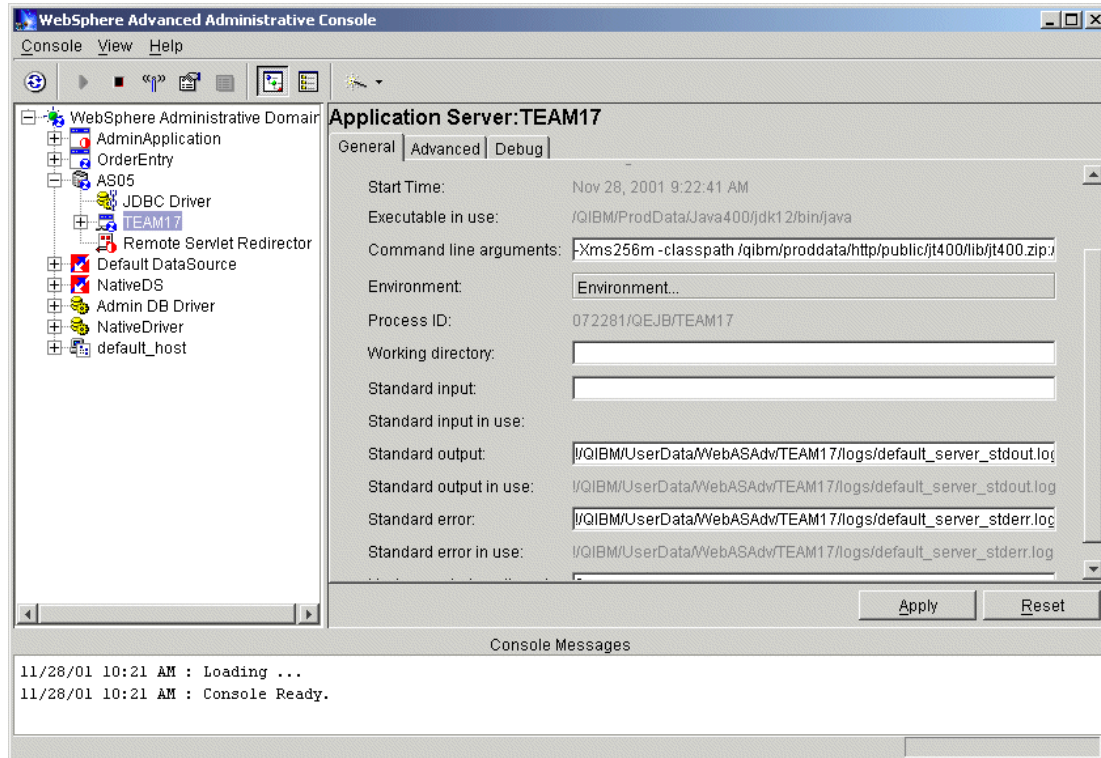


Figure 5-12 Setting the garbage collection initial size

## 5.5 Class auto reload and reload interval

Increasing the class reload interval reduces the number of times the application server scans classes for updates and reloads them. Moreover, when your application is stable and running in production, there is no need to check for changes in the servlet code, in which case you can disable the auto reload. This can reduce some overhead.

To change these two settings, follow these steps:

1. Open WebSphere Administrative Console.
2. Expand your node.
3. Expand your application server.
4. Expand **Default Servlet Engine**.
5. Click your Web application.
6. Click the **Advanced** tab.
7. Scroll down until you see the Reload Interval and Auto Reload fields (Figure 5-13) and set them according to your needs.



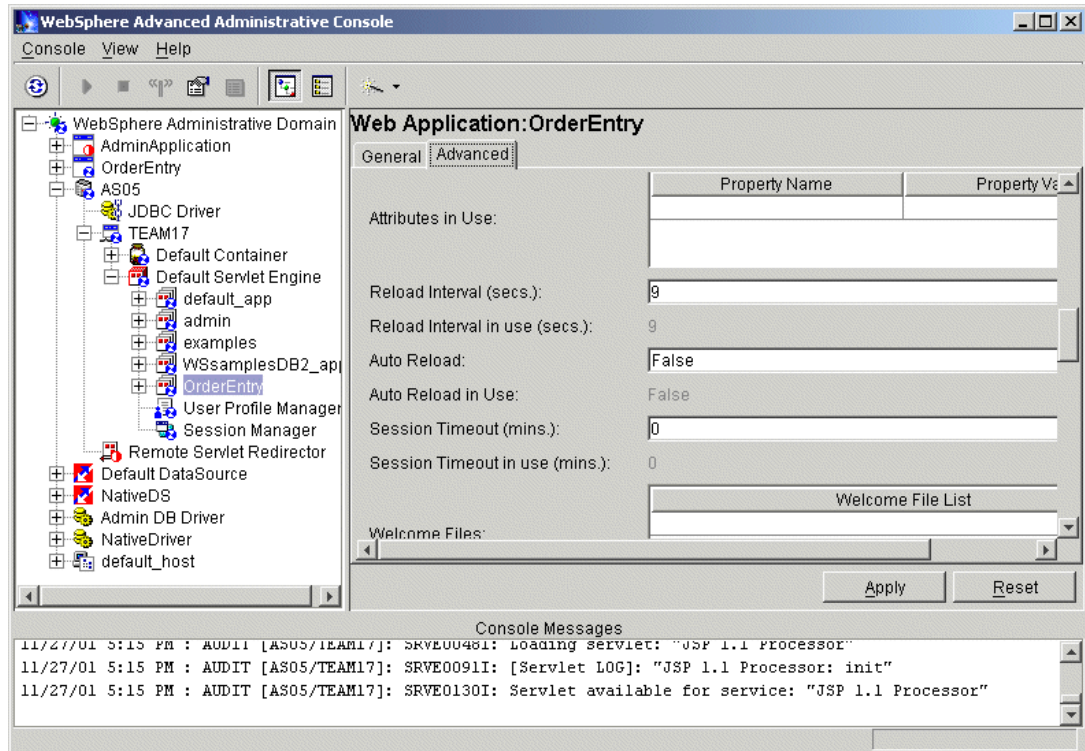


Figure 5-13 Disabling auto reload

## 5.6 Ping timeout and ping initial timeout

If you experience many application server restarts, increase the ping timeout and ping initial timeout. Changing the ping interval determines *when* the administration server believes the application server has stopped responding. When this occurs, the administration server will restart the application server. On a very busy system, this can be an undesired side effect, causing even further performance degradation.

To avoid this problem, increase the values by following these steps:

1. Open WebSphere Administrative Console.
2. Expand your node.
3. Click your application server.
4. Click **Advanced**.
5. Scroll down until you see the Ping timeout and Ping initial timeout fields (Figure 5-14) and set them according to your needs.

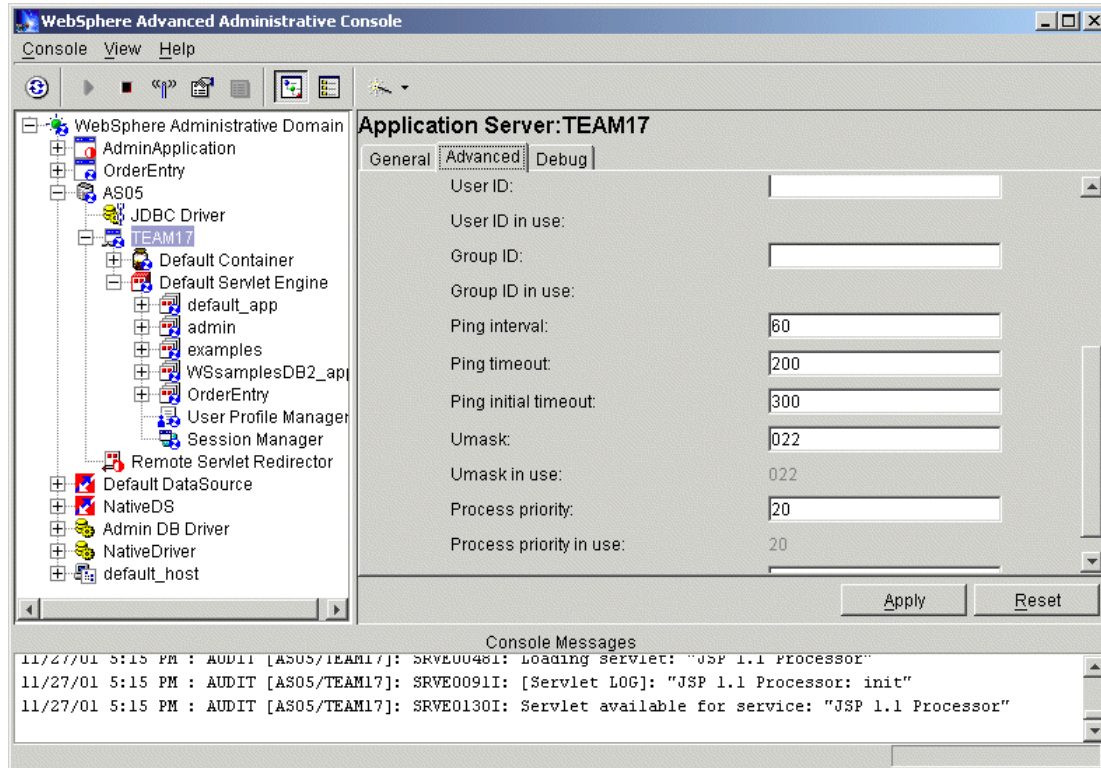


Figure 5-14 Ping interval

## 5.7 Standard output and standard error

**Attention:** This step brings only minor performance improvements, while it significantly decreases the problem determination possibilities. If a problem occurs, there will be no information available for debugging! In addition, the lack of error messages may hide a problem. For these reasons, we *do not recommend* this technique. Instead, be sure to use the technique described in 6.7.6, “Minimize or eliminate use of `System.out.println()`” on page 122.

Setting the Standard output and Standard error to "" (two double quotation marks) eliminates completely the writing of logs. This reduces some overhead because WebSphere is no longer required to synchronize around the file output. This should only be done on a stable production system where no further errors are expected.

To change these two settings, follow these steps:

1. Open WebSphere Administrative Console.
2. Click your application server.
3. Click the **General** tab.
4. Scroll down to find the Standard output and Standard error fields (Figure 5-12 on page 98). Delete the content of these fields to eliminate writing of the logs.

## 5.8 Session Manager

Use only persistent sessions if required by the application. In many cases, sessions are not required to be persisted to storage (in fact, this is mostly used for cloning). Follow these steps:

1. Open WebSphere Administrative Console.
2. Expand your node.
3. Expand your application server.
4. Click **Session Manager**.
5. Click the **Enable** tab.
6. Select **Yes** for the Enable sessions field (Figure 5-13).

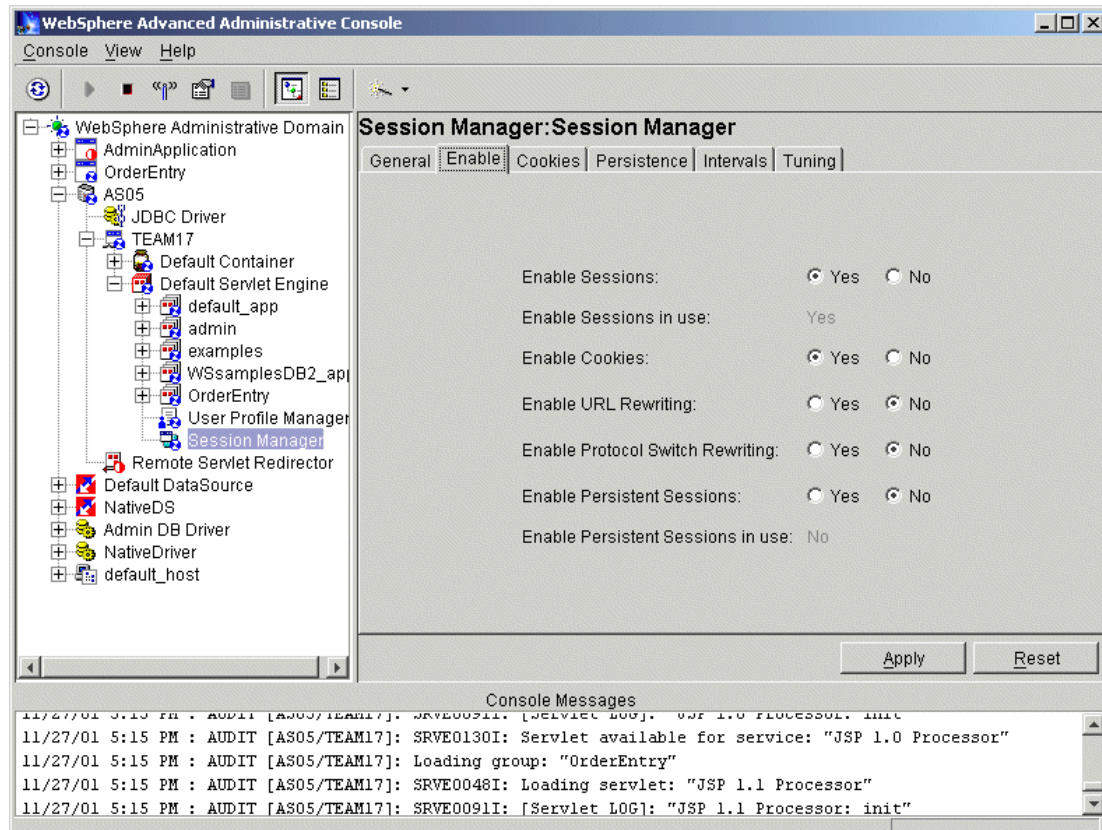


Figure 5-15 Session Manager: Enable

If persistent sessions are required, make sure you enable multi-row sessions and caching. To do this, follow these steps:

1. On the screen shown in Figure 5-15, click the **Tuning** tab.
2. Select **Yes** for the Using Multirow Sessions field (Figure 5-16).
3. Set the base memory value to a number close to the average number of active sessions. If there are frequent periods where more sessions are active, increasing the base memory value further may improve performance. To set the base memory size, specify value in the Base Memory Size field in Figure 5-16.

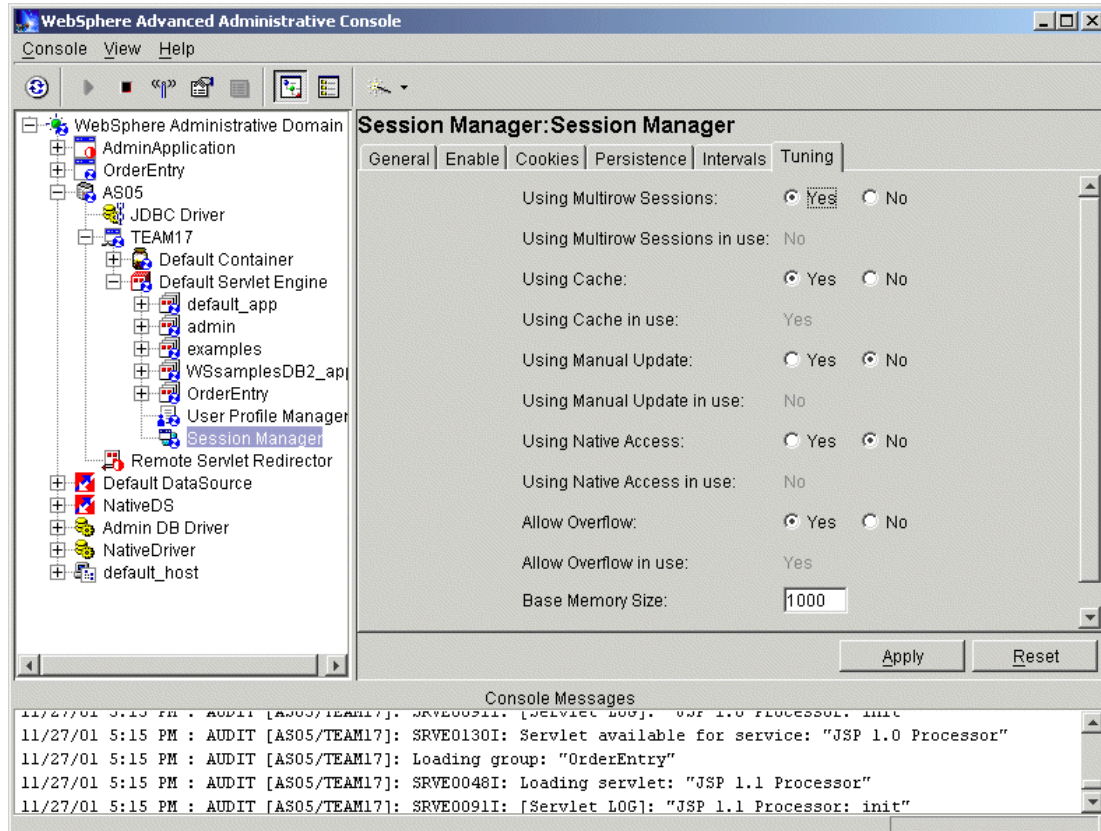


Figure 5-16 Session Manager: Tuning

## 5.9 Security

Only turn on security if it is required for the application. Enabling security has a substantial performance impact.

## 5.10 EJB container

Tune the cache settings (absolute and preferred limits, size, cleanup interval) to avoid passivation of objects.

To estimate the required value for the absolute limit property, multiply the number of entity beans active in any given transaction by the total number of concurrent transactions expected. Then add the number of active session bean instances. The WebSphere Resource Analyzer can help you obtain these measurements.

Follow these steps:

1. Open WebSphere Administrative Console.
2. Expand your node.
3. Expand your application server.
4. Click **Default Container**.
5. Click **Advanced**.
6. Set the Cache Size, Cache preferred limit, Cache absolute limit, and Cache clean-up interval fields (Figure 5-17).



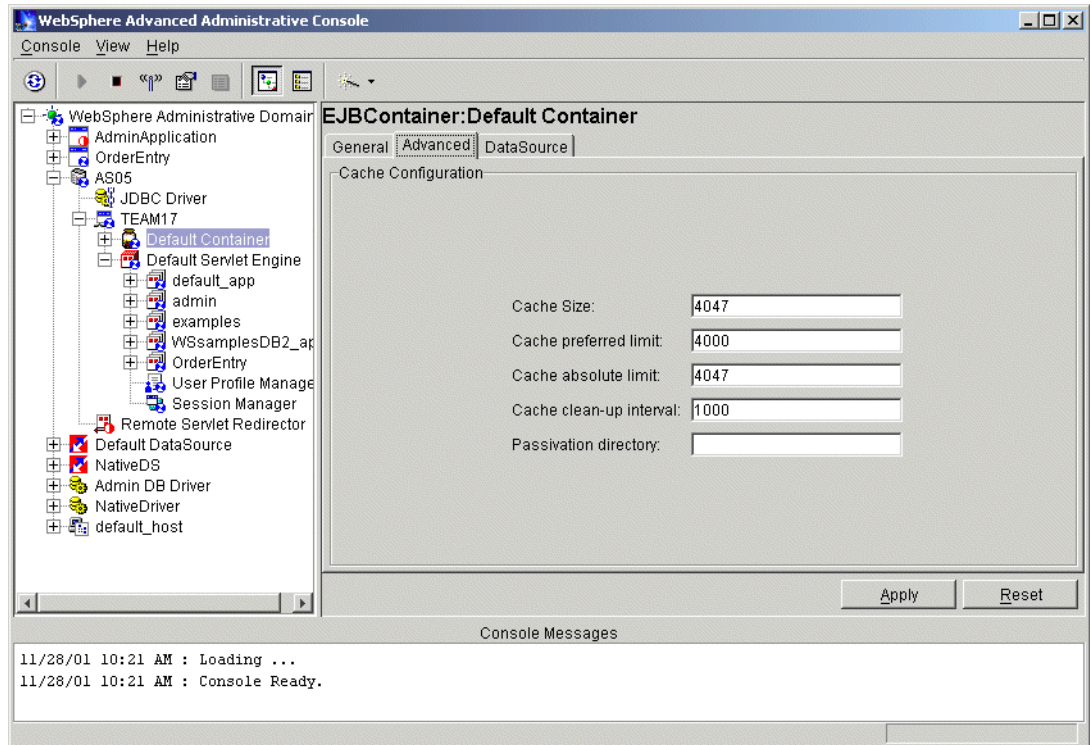


Figure 5-17 EJB container

## 5.11 Prepared statement cache

In WebSphere Application Server Version 3.5 and later, the Prepared statement cache is associated with each data source. To specify the property for setting the Prepared statement cache size, you need to create a file named *datasources.xml* and put it in your [Instance Root]/properties directory. The default location for *datasources.xml* in WebSphere Application Server - Advanced Edition is /QIBM/UserData/WebASAdv/default/properties/datasources.xml.

A sample of the contents is shown here:

```
<data-sources>
  <data-source name="Default DataSource">
    <attribute name="statementCacheSize" value="1000"/>
  </data-source>
</data-sources>
```

**Important:** The application developer will need to change the DataSource name in the XML file to match the name of the DataSource in their application. If they have multiple DataSources, they will need multiple entries in the file. To determine the proper size of the cache, multiply the number of unique statements in the application by the maximum number of concurrent database connections. In practice, setting it to an arbitrarily large value, such as 1000, tends to work just fine.

## 5.12 WebSphere plug-in to HTTP server

WebSphere administration tracks a variety of configuration information about WebSphere resources. Some of this information needs to be understood by the HTTP server as well, such as uniform resource identifiers (URLs) pointing to WebSphere resources.

In WebSphere Application Server Version 3.5 and earlier, the configuration data is pushed to the HTTP server via the WebSphere plug-in in regular time intervals. This allows new servlet definitions to be added without having to restart any of the WebSphere servers. Unfortunately, the dynamic regeneration of this configuration information is an expensive operation. You can boost performance by increasing the configuration update interval and, therefore, make configuration updates run less often.

**Note:** In WebSphere Application Server Version 4, the configuration data is not updated regularly. You need to restart your application server in order to update the configuration.

WebSphere Application Server Version 3.5 has a setting in `[webspheredroot]/properties/bootstrap.properties` that is used to change the refresh interval between these updates. The property name is `ose.refresh.interval`. The default value for `ose.refresh.interval` is 10. Changing this number to a higher value may improve performance.

## 5.13 JSP processing engine

You can reduce the update checking frequency by adding an `Init` parameter to the JSP servlet (JSP 1.0 and above). In WebSphere 3.02 and above, the JSP processing engine is implemented as a servlet. The JSP 1.0 servlet is invoked to handle requests destined to all JSP files matching the assigned URL filter, such as `/myApp/*.jsp`. Each time a JSP file is requested, the corresponding disk file is checked to see if a newer version is available. This can be an expensive operation.

Change the interval by adding an `Init` parameter to the JSP Servlet. The parameter name is *minTimeBetweenStat*, with a default value of 1000, although 100000 is recommended. Then, follow these steps:

1. Open the WebSphere Administrative Console.
2. Expand the node.
3. Expand your application server.
4. Expand your Web application.
5. Click **JSP 1.1 Processor**.
6. Click the **Advanced** tab (Figure 5-18).

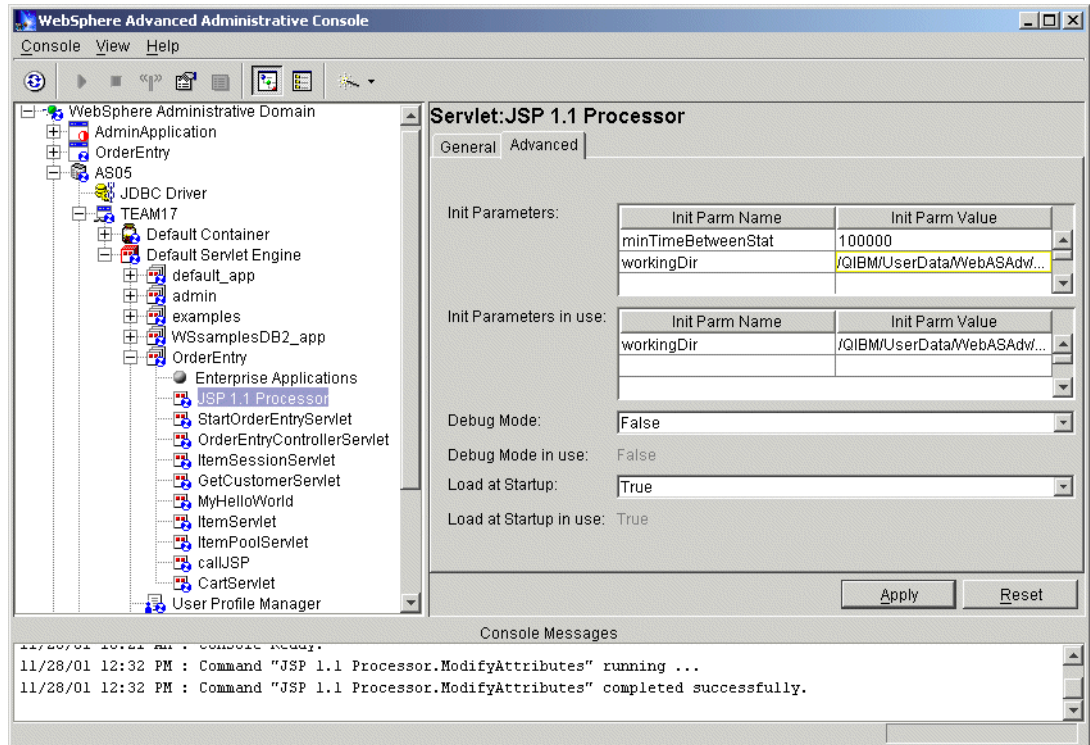


Figure 5-18 JSP processor Init parameter

## 5.14 Tuning parameters to leave untouched

As you explore the WebSphere Administrator Console looking for performance opportunities, you may come across some of the settings covered in this section. These are the settings that you need to leave alone on the iSeries server because altering them may have an adverse performance impact (or at least offer no benefit).

### Trace and debug

Do not enable tracing unless required for debugging. Tracing adds costly overhead to your application execution and should never be enabled in a production environment.

To check that tracing is off, follow these steps:

1. Open your WebSphere Administrative Console.
2. Expand the node.
3. Expand your application server.
4. Click **Advanced** and make sure the Trace Specification field is empty (Figure 5-19).

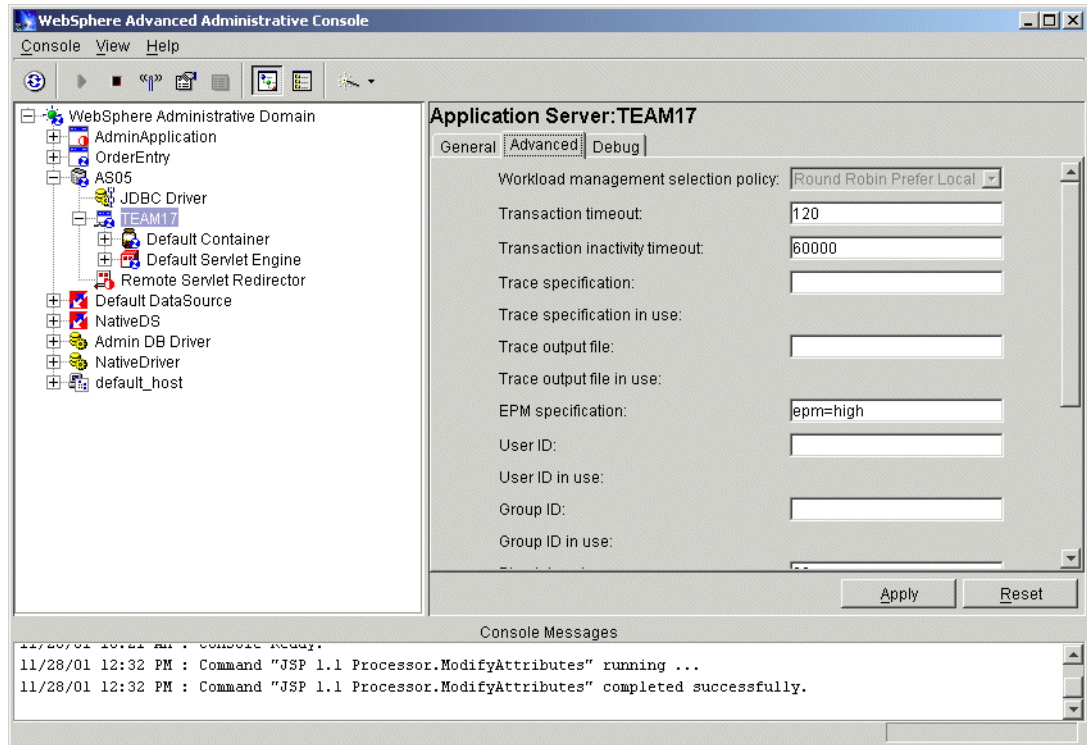


Figure 5-19 Disabling trace

- Click the **Advanced** tab and make sure the **Debug enabled** and **Object Level Tracing enabled** check boxes are deselected (Figure 5-20).

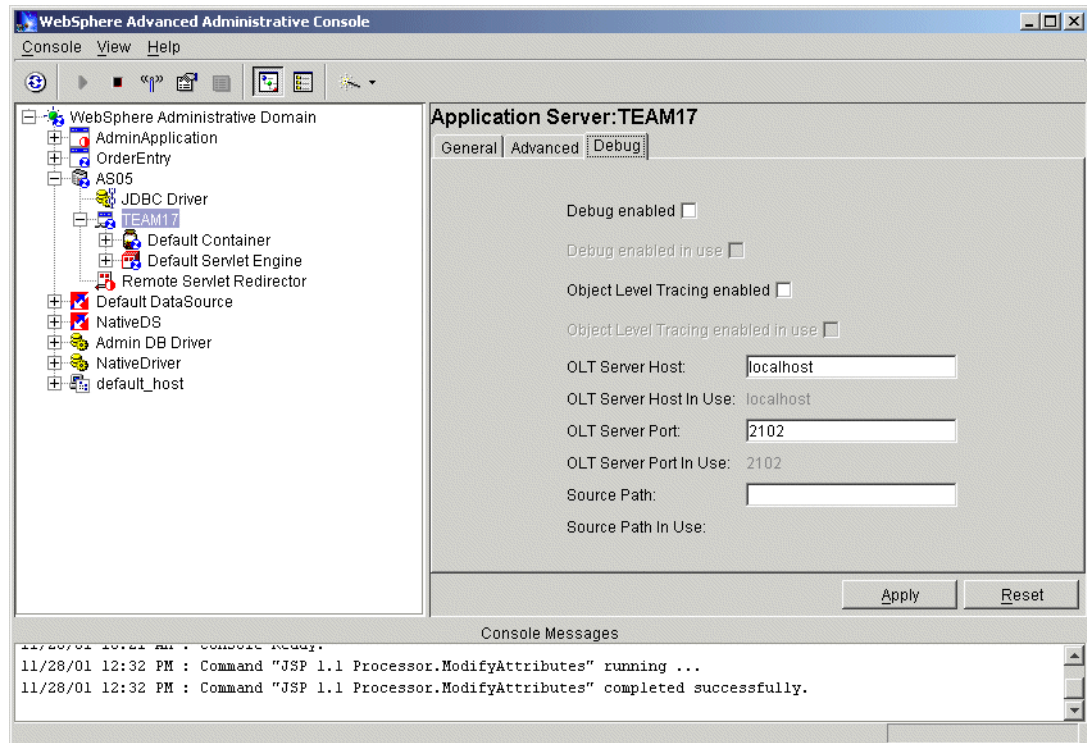


Figure 5-20 Disabling debug



## Extended Performance Monitoring (EPM) specification

Unless you are monitoring WebSphere Application Server for performance (for example, using the WebSphere Resource Analyzer), do not enable EPM because it activates a large number of internal performance counters. To make sure EPM is disabled, follow these steps:

1. Open the Administrative Console.
2. Expand the node.
3. Click your application server.
4. Click the **Advanced** tab.
5. Make sure the EPM specification field is set to none (Figure 5-21).

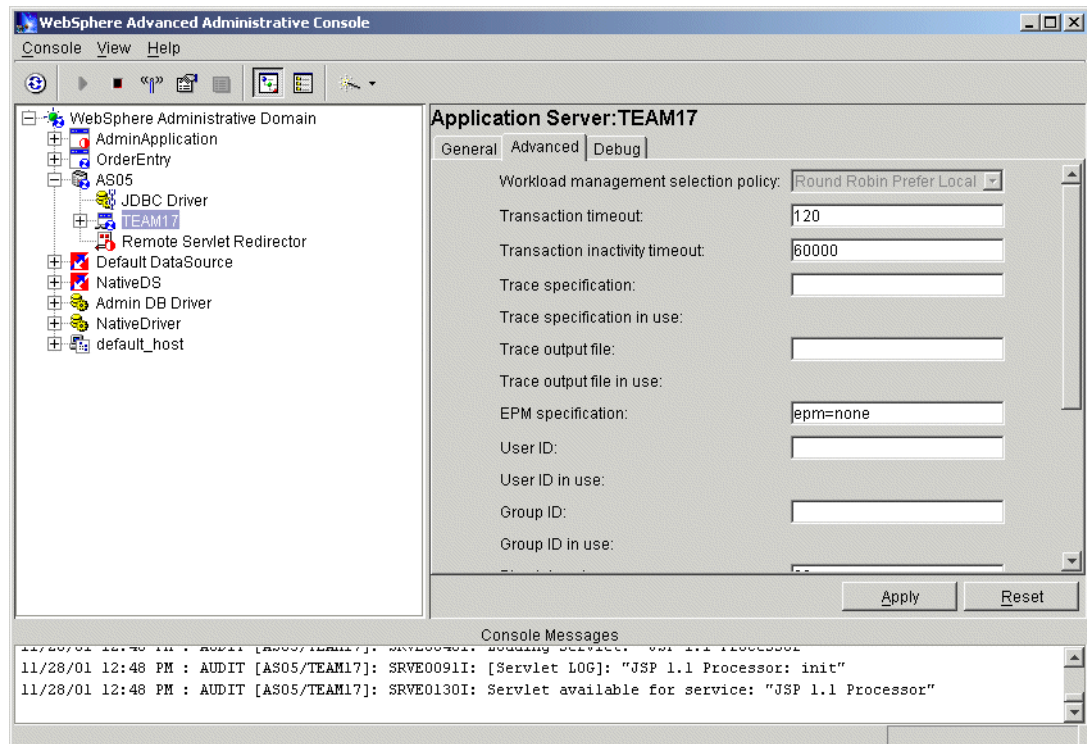


Figure 5-21 Disabling EPM

## Ping interval

Leave the ping interval at its default value. Do not change the ping interval, because this could interfere with the time-out values and cause unexpected restarts. Make sure it is set to the default value of 60 (Figure 5-14 on page 100).

## ‘No Local Copies’

Because EJBs are inherently location independent, they use a remote programming model. Method parameters and return values are serialized over RMI-IIOP and returned by value. This is the intrinsic RMI “Call By Value” model. WebSphere provides the “No Local Copies” performance optimization for running EJBs and clients (typically servlets) in the same application server JVM.

The “No Local Copies” option uses “Call By Reference” and does not create local proxies for called objects when both the client and the remote object are in the same process. Depending on your workload, this can result in a significant overhead savings.

You enable “No Local Copies” by adding the following two parameters to the command line options in the WebSphere Administrative Console:

```
-Djavax.rmi.CORBA.UtilClass=com.ibm.CORBA.iiop.Util  
-Dcom.ibm.CORBA.iiop.noLocalCopies=true
```

**Important:** The “No Local Copies” configuration option improves performance by changing “Call By Value” to “Call By Reference” for clients and EJBs in the same JVM. One side effect of this is that the Java object derived (non-primitive) method parameters can actually be changed by the called enterprise bean.

Test your application carefully before you enable this feature on a production system.

### Settings that are ignored on the iSeries server

The following settings may look like interesting options to help improve performance, but they are ignored on the iSeries server:

- ▶ Process priority
- ▶ Thread pool size

### Servlet engine settings

WebSphere Application Server Version 3.5 uses the OSE communication protocol for internal purposes as well as to communicate with HTTP server. In WebSphere Application Server Version 4, OSE has been replaced with HTTP.

**Attention:** This section applies only to WebSphere Application Server Version 3.5.

Leave OSE as the default queue type for the servlet engine. Do not attempt to change the queue type to HTTP because this will slow performance.

In the Administrative Console, follow these steps:

1. Expand the node.
2. Expand your application server.
3. Click **Default Servlet Engine**.
4. Click the **Advanced** tab.
5. Make sure the Queue Type field is set to **OSE**.

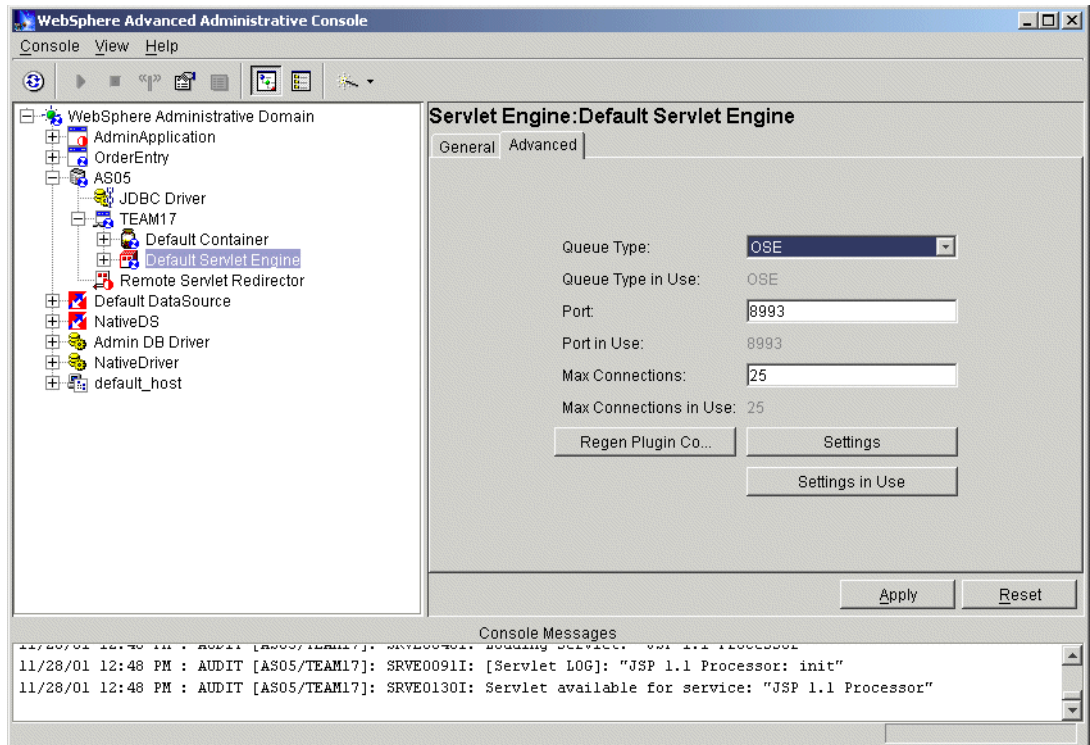


Figure 5-22 Servlet engine queue

- Click the **Settings** button and make sure **Local Pipes** is the default transport type when using OSE (Figure 5-23). Do not attempt to use INET or TCP/IP because this will slow performance.

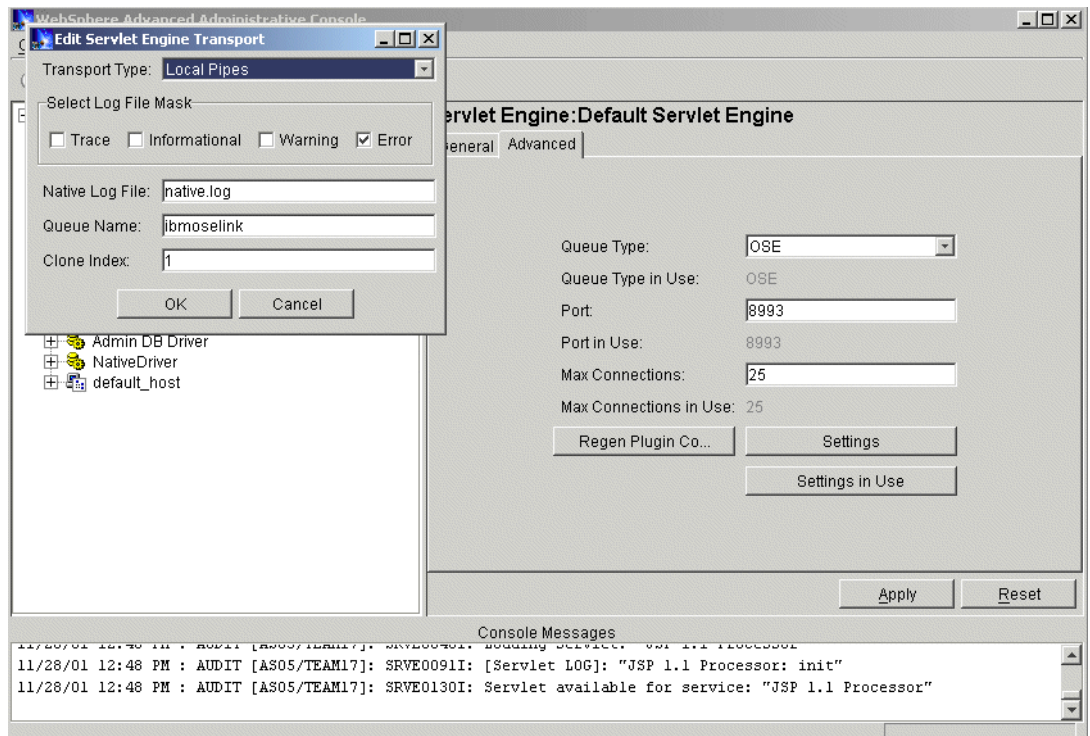


Figure 5-23 Local pipes





## Java and WebSphere application design

The area where most performance can be significantly improved is usually the application itself. This chapter explains the relationship between Java application design and host performance. It starts with some of the basic rules for Java development and JDBC access. Then it moves on to discuss some additional best practices that apply to coding and deploying applications for the WebSphere Application Server.

## 6.1 Java design

Because Java is a portable language, you hope that the coding techniques that work well on one platform are also beneficial on other platforms. In many ways, this is true. Even before the advent of Java, seemingly dissimilar systems had certain similarities in terms of what operations tended to be very expensive. These specifically pertain to any operations that required the creation and destruction of anything. For example, the creation and deletion of a file or table, the creation and destruction (initiation and termination) of a job, the opening and closing of a file (results in creation and destruction of an Open Data Path), are all relatively expensive. This chapter discusses certain techniques that are available in the Java environment that tend to result in the instantiation and deletion of objects. Fortunately, there are alternatives to minimize object creation and deletion.

Application design primarily affects the areas of Java instruction execution and access to system services. The subject of application design has broad coverage. This section presents design and coding considerations that apply to Java runtime performance and various recommendations for accessing DB2 UDB for iSeries. Note that since Java on the iSeries system plays a significant role in the e-business environment, access to the database has a much larger significance than in non-commercial applications.

## 6.2 Instruction execution: Coding considerations

There are certain programming techniques that result in high amounts of overhead. These techniques appear to be costly because they result in object creation and deletion, as well as other overhead that is inherent to an object-oriented design. In the relatively early stages of Java application development, coding is generally done with functionality in mind. Performance is generally an afterthought. However, certain programming techniques have more overhead than desired and, therefore, should be minimized. It is also possible to carry something to an extreme, for example, creating tens of thousands of objects when one is sufficient.

### 6.2.1 Instantiation results in garbage collection

*Instantiation* is the creation of new objects. This is expensive because it results in the allocation of heap storage. As expected, instantiation eventually results in garbage collection since the objects that were created eventually have to be cleaned up when they are no longer referenced. The more complex an object is, the more overhead it takes to instantiate it.

Instead of creating a new instance of an object each time with the "new" operation, it may be possible to reuse an object by varying its attributes or instance variables. Obviously, unnecessary object instantiation needs to be eliminated. The use of lazy initialization techniques to determine if an object needs to be created is more efficient:

```
if (securityManager == null)
    securityManager = new RMISecurityManager;
```

This way, the securityManager object is created only if it does not already exist.

## 6.2.2 String manipulation

The problem with string manipulation is that, in Java, strings are immutable. This simply means that the Java specifications dictate that strings are not capable of change. Therefore, changes to the value of a string (for example, assignments or concatenation) result in the old string object being discarded and a new string object being created to contain the new value. String manipulation can then result in large amounts of object instantiation and garbage collection.

A technique for reducing this overhead is to use a `StringBuffer` object instead of a string. The `StringBuffer` is not immutable and has several methods, one of which is `StringBuffer.append`. Appending to a `StringBuffer` is a much better performer than string concatenation. However, being capable of change means that there is a corresponding synchronization overhead associated with this object type. An even faster technique for variable string values is to use an array of characters (`char[]`) to behave like a fixed-length string.

## 6.2.3 Method resolution

Properly designed object-oriented applications are much more modular than procedural implementations. This results in more frequent method resolution, or the invocation of methods that do not exist in the class that performed the request. Instead, the methods can represent behaviors that are inherited from a super class. When the method is called, the program searches for it within the current class. If it is not found, the method is searched for externally. Then, the external method is resolved and invoked, with the expected arguments passed to the external method. Obviously, this process is more expensive than if the method was located in the same class that called it.

One way to reduce this is to specify methods as `final`, when possible, and then to compile them with the `javac -O` option. This causes *method inlining*. This means that the method call is replaced with the actual method instructions (bytecode) being copied into the subclass that requests it. This eliminates the need for the application to resolve or search for the method and to access the external method. This can result in substantial runtime improvements if method resolution is a large portion of the application overhead. Note, however, that this will result in larger class sizes for the affected classes because a copy of the method's bytecode is now embedded within the class. Since V4R4, creating a Java program under `OPTIMIZE(40)` performs method resolution without requiring the `javac -O` option.

## 6.2.4 Exception handling

Using exception handling mechanisms, such as "throws exception", is relatively expensive. You should not use exception handling mechanisms for regular processing. They should be used only in real exception conditions.

Try-catch structures are very efficient. It is far more efficient to catch the occasional exception than to pass return codes and evaluate them.

## 6.2.5 Java Native Interface (JNI)

The implementation of the JVM is fully integrated into the OS/400 Machine Interface. Frequent accesses of trivial non-Java programs through JNI should be minimized since this requires processing above the Machine Interface and tends to slow the application down.

As of V4R4, a safe JNI call can be made to ILE RPG and ILE COBOL programs since they were made thread-safe. Previously, only C programs were recommended for JNI calls.

However, calls to longer running non-Java programs may be tolerable because the overhead of making the call is small compared to the total amount of productive work done.

### 6.2.6 Thread creation

The iSeries system kernel thread implementation is very efficient. Thread creation is considered a lightweight operation. Threads within a process can re-use heap pages among themselves. However, extreme designs have also been observed, for example, one process with tens of thousands of threads. Although the iSeries server is more than capable of handling such designs due to its multi-tasking heritage, unnecessary thread creation simply adds up to additional overhead.

One area where threads can really be useful is in reducing the perceived response time of a transaction. Instead of serially processing a long running transaction, other threads can be spawned to multi-task disparate parts of the transaction.

### 6.2.7 Variable scope

The scope of variables has different costs. Access to a local variable is the fastest. Direct access to an instance variable or using an inlined accessor method have approximately the same cost. Both are more expensive than local variable access. Using an accessor method (not in-lined) is even more expensive. The most expensive method is to use a synchronized accessor method.

### 6.2.8 Remote AWT

Because the iSeries server is not designed to generate and display graphics, method calls to AWT or Swing methods are routed to a graphical client where they belong. This is done through an iSeries mechanism called *Remote AWT*.

In V4R2, Remote AWT was implemented internally through Remote Method Invocation (RMI). Significant performance improvements were attained in V4R3 when the RMI layer was removed and a direct sockets connection was used. Additional enhancements were made for V4R4.

Remote AWT is not designed for high-volume transaction processing. Think about the amount of traffic that has to flow through the network, including events such as mouse moves, mouse clicks, and so on. The GUI portion of an application should run on the client where it belongs, not on the server to be handled by Remote AWT.

However, for low-volume transactions, such as installing an off-the-shelf package, the network traffic is not as significant so Remote AWT usage is not a problem.

## 6.3 Accessing system services: Database operations

In an typical commercial environment, there are two major areas where processing overhead is incurred. One area is in running the Java program instructions, as described in the previous section. The other is in accessing the system services. In most cases, the majority of the system services are represented by database operations.



Some of the recommendations implemented herein are specific to the iSeries server platform and may not apply to other servers. For example, record level access through Distributed Data Management (DDM) and Distributed Program Call (DPC) are done through the IBM Toolbox for Java and apply only to the iSeries server. Other recommendations pertaining to Java Database Connectivity (JDBC) may apply to other relational databases.

Figure 6-1 shows various ways to access the relational database or existing programs on the iSeries server.

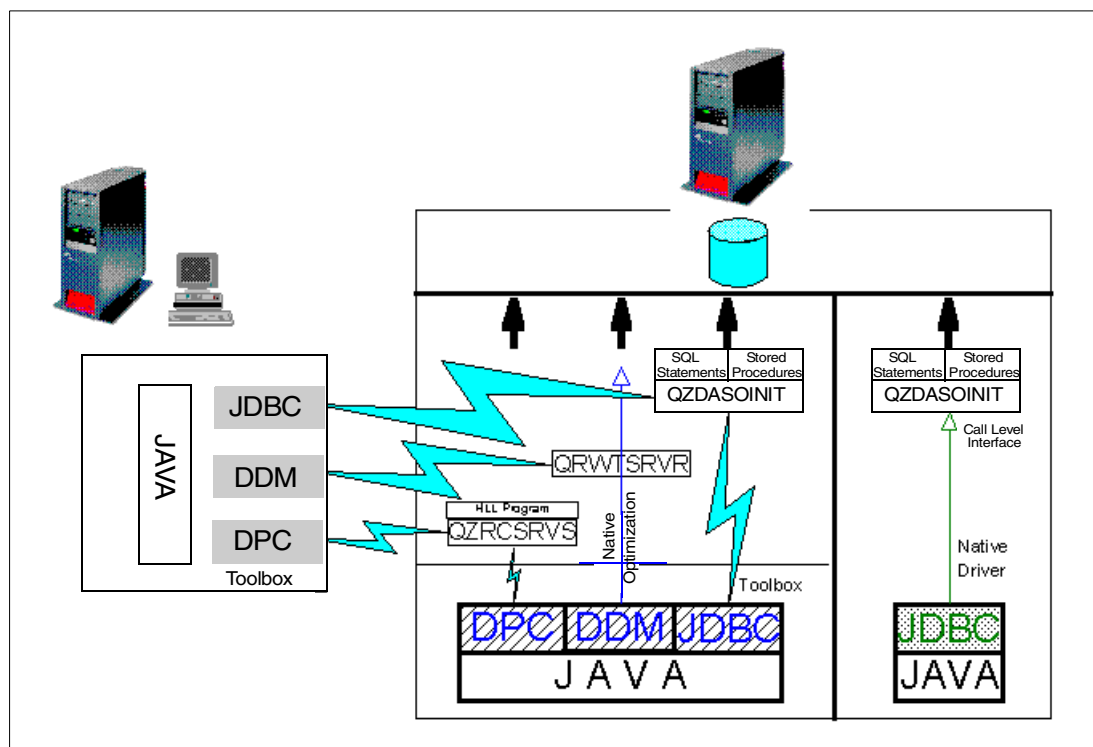


Figure 6-1 IBM Toolbox for Java system access

In Figure 6-1, the left portion represents a Java application running on a client system and using the IBM Toolbox for Java classes to access the iSeries server. Since the IBM Toolbox for Java classes are 100% pure Java, this application is portable to any compliant JVM. However, the server that it accesses has to be an iSeries server.

The middle portion represents a Java application running on the iSeries server using various ways to access system services. These include:

- ▶ The DPC model is designed for client-server work so the Java application communicates through the QZRCRSVS server job to access an iSeries program.
- ▶ The DDM or record-level access model has the IBM Toolbox for Java classes "optimized for native". This means that if the IBM Toolbox for Java classes determine that access is being done locally, there is no need to go through a server job or through sockets. The Java application accesses the database directly.
- ▶ This JDBC approach uses the IBM Toolbox for Java JDBC driver.

The right portion in Figure 6-1 shows a server Java application using the "native" JDBC driver.

Regardless of which driver is used, JDBC can access DB2 through SQL statements or through stored procedures. The iSeries server is unique in the industry because stored procedures do not have to contain embedded SQL statements. Even traditional 3GL programs using native I/O operations can be used as stored procedures.

### 6.3.1 IBM Toolbox for Java driver versus iSeries Developer Kit for Java driver

As described previously, there are two JDBC drivers available when running a Java application on the iSeries server. One is the native driver or the iSeries Developer Kit for Java driver, `com.ibm.db2.jdbc.app.DB2Driver`. The other is the IBM Toolbox for Java driver, `com.ibm.as400.access.AS400JDBCDriver`. Some of the properties may vary, but the SQL functionality is mostly similar. The IBM Toolbox for Java driver is a type 4 driver and provides network access for clients that are physically separated from the database. The iSeries Developer Kit for Java driver is a type 2 driver that provides direct access.

Other differences are described in *The iSeries Developer Kit for Java*. From a performance standpoint, there are more significant differences. You can find *The iSeries Developer Kit for Java* in the iSeries Information Center at:

<http://publib.boulder.ibm.com/pubs/html/as400/infocenter.html>

#### IBM Toolbox for Java driver performance

The IBM Toolbox for Java driver was originally designed to let a client access the database in a client/server environment through a network. It uses a sockets connection between the client and the server. The server job to which the JDBC requests are submitted is named QZDASOINIT. This is the same server job name used to process 32-bit client requests in the ODBC environment.

The IBM Toolbox for Java driver allows SQL extended dynamic support, which allows SQL statements and their access plans to be stored in objects of type SQL Package (\*SQLPKG) on the iSeries server. This provides a significant performance benefit for repeatable SQL statements because of the reduced parsing and syntax checking overhead. One performance difference between the IBM Toolbox for Java JDBC environment and ODBC is that in the latter, Extended Dynamic mode is false, by default.

However, when used in a server Java application within the same iSeries server that contains the data, the current implementation does not "short circuit" the sockets implementation. This means that requests from a server Java program go through the sockets interface before accessing the database. The result is additional latency as well as additional processing overhead.

#### iSeries Developer Kit for Java driver performance

The iSeries Developer Kit for Java driver, on the other hand, was designed to process SQL requests from a server Java program to access DB2 UDB for iSeries. It uses the Call Level Interface (CLI) standard to submit SQL requests. CLI is like ODBC running natively on the iSeries server, with essentially the same APIs. It produces dynamic SQL statements, as do the ODBC or Toolbox JDBC environments. The difference is in the way that SQL statements are cached.

Past experience with dynamic SQL in various computing platforms brings back memories of poor performance. Most of the problem was due to statement parsing and syntax checking. However, starting with OS/400 V4R2, an internal system-wide SQL statement cache was implemented, which significantly improves the execution of dynamic SQL. This cache stores dynamic SQL statements for much improved subsequent execution.

Currently, there is a maximum of 500 concurrent statement handles per connection. In the unlikely situation that more than 500 are required, another connection can be established.

### **Which one to use**

When running Java on the iSeries server, we recommend that you use the iSeries Developer Kit for Java driver or a "native" driver, both for server efficiency and total response time.

Use the IBM Toolbox for Java driver when issuing requests from a client to the server.

Switching from one driver to another is a simple change, usually affecting only one class. Using one over the other in the server application does not mean that the application needs to be redesigned if the driver needs to be switched. A best practice is to keep the name of the JDBC driver and its properties in an external property file. This simplifies portability and avoids any code changes due to driver change.

## **6.3.2 JDBC through a native driver**

When a Java program on the iSeries server performs SQL requests to DB2 UDB for iSeries, the iSeries Developer Kit for Java JDBC driver issues dynamic SQL requests through the CLI standard. Under the CLI implementation, the SQL requests are not performed by the Batch Immediate (BCI) job. They are sent to a separate pre-started job named QSQSRVR. When Java establishes a connection to the DB, a message is sent to the BCI job log:

```
123456/QUSER/QSQSRVR used for SQL server mode processing.
```

When you analyze the database, monitor the QSQSRVR job for optimizer messages. This is done by placing the job in service mode and debug mode (using the STRSRVJOB command followed by the STRDBG command). Its job log then records the optimizer messages.

The Database Monitor can also be used to monitor this job (using the STRDBMON and ENDDBMON or the STRPFRMON commands). This monitor provides much more information than when observing the job log in debug mode. Actual SQL statements are recorded.

Another option for analyzing SQL requests is the SQL Monitor available in V4R4 Operations Navigator.

### **Statement cache for dynamic SQL**

There are three kinds of SQL statements that can be issued on the iSeries server:

- ▶ Static SQL
- ▶ Dynamic SQL
- ▶ Extended dynamic SQL

Static SQL is the most efficient, but is currently not available to a server Java application unless it is issued from a stored procedure. Future Java releases may provide this option in the form of SQLJ. However, current implementations of SQLJ provide "ease of coding" advantages rather than performance advantages.

Static SQL, as the term indicates, is assumed to be constant. The statements and the access plans are stored in the program object that contains the embedded SQL statements. These are still the best performing SQL statements because there is no need to parse, check the syntax, and possibly create the access plan for every execution of the statement.

Extended dynamic SQL (strictly speaking, these can be considered a form of dynamic SQL statements), when issued from a client, has a fairly good performance profile, close to that of static SQL. The reason is that SQL statements are stored in an SQL package (\*SQLPKG) object. However, as mentioned earlier, the IBM Toolbox for Java JDBC driver is not optimized for requests from Java applications within the server even though it generates extended dynamic.

Starting with V4R2, an SQL statement cache is implemented internally. This statement cache stores dynamic SQL statements so that subsequent executions of the same statement run much faster. Since this is a system-wide cache, a statement that is executed in one job also benefits other jobs. Using the statement cache is automatic and is available to all applications that generate dynamic SQL, not just server JDBC requests. You don't need to do anything to activate it. The SQL statement cache is cleared at IPL time.

Unlike other platforms where the SQL statement has to be an exact match (including spaces), the iSeries cache is more tolerant of differences and can also handle parameter marker replacement. Not only are the statement strings stored, but other internal structures such as access plans are stored as well, reducing the need to recreate them each time the statement executes.

## **Database tuning**

In an online transaction processing environment (OLTP), the query runtime should be very short. The database should then be tuned properly. Tables should have the proper indexes, and the SQL statements should be written properly to use these indexes. The proper level of normalization and the use of the SMP feature also helps.

In analyzing JDBC environments, the debug feature or DB Monitor should be used to determine the amount of time spent in running the queries. These facilities also provide recommendations on which indexes need to be created.

Other ways to monitor and analyze optimizer messages are to put the SQL server job in debug mode and the SQL Monitor within Operations Navigator.

Proper database tuning also improves the probability of the application generating reusable Open Data Paths (ODPs). Generally, using temporary objects, such as indexes or tables, results in non-reusable ODPs.

## **Stored procedures CLOSQLCSR**

For stored procedures that are called repeatedly within the same connection, you should keep the ODPs open by specifying the proper CLOSQLCSR parameter such as \*ENDJOB or \*ENDACTGRP.

# **6.4 Coding considerations with JDBC**

This section covers some of the generic issues that to consider when coding programs using JDBC.

## **6.4.1 Data conversion**

Database servers generally store data using a different encoding scheme from Java. The iSeries JVM stores string data as 2 byte Unicode. If the database is encoded in EBCDIC, the data will be converted on every database access. You can avoid this by storing the data in DB2 as 2 byte Unicode, using the SQL graphic type with CCSID 13488 or the DDS graphic type with CCSID 13488.

Store numeric data in DB2 as a float to reduce numeric conversions. Decimal data cannot be represented in Java as a primitive type. Decimal data is converted to the abstract class *Java.lang.BigDecimal* on every database read or write. This conversion can be avoided by storing numeric data in the database as float or double. Alternatively, you may consider converting the *BigDecimal* type to float for heavy calculations while leaving the database in decimal form. Note that rounding differences may be introduced with the use of float or double.

## 6.4.2 Retrieve only the necessary columns

It is very common to write SQL statements making use of the syntax:

```
select * from employees where....
```

This has the side effect of returning every single column in the file in the result set. A much faster and, in effect, just as easy technique to use is to select the specific columns needed using the following syntax:

```
select firstname, lastname, department from employees where...
```

This only brings back the necessary columns. It can also reduce the number of EBCDIC to Unicode conversions taking place if the data is stored in EBCDIC.

## 6.4.3 Use get...(columnIndex) instead of get...(columnName)

“Getting” column data by index instead of name is faster, since the JDBC driver does not have to perform name resolution. Note that on some platforms, using a column index requires the gets to be performed in column sequence. This is not a restriction on the iSeries server.

## 6.4.4 Avoid using AutoCommit

Using *AutoCommit* generally results in a very large number of commits, which may not be required. You achieve better performance (and often also transaction integrity) by managing your own commits.

## 6.4.5 Consider using batch insert

The batch update facility allows a *Statement* object to submit a set of heterogeneous update commands together as a single unit, or batch, to the underlying DB2 database. This is a concept designed purely to boost performance.

In the example in Figure 6-2, *AutoCommit* mode is disabled to prevent JDBC from committing the transaction when *Statement.executeBatch()* is called. Disabling *AutoCommit* allows the application to decide whether to commit the transaction in the event that an error occurs and some of the commands in a batch fail to execute. For this reason, *AutoCommit* should usually be turned off when batch updates are done.

On the iSeries server, using batch insert can yield up to 10 times the performance, but beware of flooding the JDBC cache.

```
// Turn off auto commit
con.setAutoCommit(false);
Statement stmt = con.createStatement();
stmt.addBatch("INSERT INTO employees VALUES (1000, 'Joe Jones')");
stmt.addBatch("INSERT INTO departments VALUES (260, 'Shoe')");
stmt.addBatch("INSERT INTO emp_dept VALUES (1000, 260)");
// submit a batch of update commands for execution
int[] updateCounts = stmt.executeBatch();
con.commit();
```

Figure 6-2 Using batch insert

### 6.4.6 Use prepared statements

Even with the statement cache, it is still necessary for performance reasons to prepare SQL statements (`prepareStatement()`) prior to executing (`executeQuery` or `executeUpdate` methods) them repeatedly. If the repeatable statement is not prepared beforehand, the `executeQuery` causes an implicit `prepareStatement` to occur every time the statement is run. Therefore, the overhead will be higher without an explicit prepare.

You prepare the statement once using the `prepareStatement()` method on the `Connection` object and then execute it whenever required using `execute()`, `executeQuery()`, or `executeUpdate()` on the `PreparedStatement` object.

## 6.5 Coding considerations with DDM (record level access)

There is a significant difference between SQL access and record level access. In the former, the programmer specifies what to retrieve. Then, the optimizer decides how to retrieve it. Where in the latter, the programmer decides how and what to retrieve. Therefore, efficient data retrieval logic is important in record level access. This problem is not new in Java. It has persisted since the first RPG, COBOL, or C programs were written.

A common scenario is reading multiple records and checking a status flag until the required one is found. It is much more efficient to include such logic in the database, for example, as an index with the proper selection criteria. That way, only the needed record is read.

### 6.5.1 Minimizing open and close

Repeated execution of the `open()` and `close()` methods over the same file is expensive and unnecessary. The program logic should be implemented so that a file is opened only once and kept open within the Java application.

### 6.5.2 Blocking on READ\_ONLY and WRITE\_ONLY

A file that is opened for input-only can take advantage of blocked input when it is appropriate. Note the following example:

```
myKeyedFile.open(AS400File.READ_ONLY, 100, AS400File.COMMIT_LOCK_LEVEL_NONE);
```

The attribute of `100` is the blocking factor, which means that the first `read()` operation retrieves as many records in a single operation. Subsequent invocations of `read()` retrieves data from the buffer instead of going back to the database. A similar benefit can be derived from a file that is opened with a `WRITE_ONLY` attribute.

### 6.5.3 Downloading a small file using the readAll() method

When retrieving the data from a small file, the `readAll()` method is more efficient. All of the records in the file are retrieved in one method call. The client should have enough resources to contain the entire file.

## 6.6 Distributed Program Call (DPC)

DPC may be used to call to existing RPG, COBOL, or C programs. A typical performance pitfall with such languages is that the programs and their files are usually closed down when they are exited. For a repeatedly called RPG program, the LR indicator should not be set on.

For OPM COBOL, it may not be possible to keep the programs and files open. Running an `EXIT PROGRAM` or `GOBACK` from a COBOL main program ends the run unit, shutting everything down. However, the ILE COBOL/400 environment will perform much better.

In a recent release, an IBM extension was added to the `EXIT PROGRAM` operation. The `EXIT PROGRAM AND CONTINUE RUN UNIT` clause allows the COBOL run unit to remain active. It keeps the files and programs open for subsequent calls.

## 6.7 Coding considerations for servlets and JSPs

All the coding considerations shown previously of course also applies to coding servlets and JSPs. There are, however, a number of coding conventions specific to coding for a servlet engine. This section covers some of these.

### 6.7.1 Do not store large objects in HttpSession

Large applications require using persistent `HttpSessions`. However, there is a cost. An `HttpSession` must be read by the servlet whenever it is used and rewritten whenever it is updated. This involves serializing the data and reading it from and writing it to a database.

In most applications, each servlet requires only a fraction of the total session data. However, by storing the data in the `HttpSession` as one large object graph, an application forces WebSphere Application Server to process the entire `HttpSession` object each time.

WebSphere Application Server has `HttpSession` configuration options that can optimize the performance impact of using persistent `HttpSessions`. The `HttpSession` configuration options are discussed in detail in the documentation that comes with WebSphere Application Server. Also consider alternatives to storing the entire servlet state data object graph in the `HttpSession`, such as using your own JDBC connection alternative.

When configuring persistent sessions in WebSphere Application Server, use a dedicated data source. To avoid contention for JDBC connections, don't reuse an application `DataSource` or the WebSphere Application Server repository for persistent session data.

As an alternative to storing the entire object graph in the `HttpSession`, use JDBC for partitioning and maintaining the state data needed by each servlet in the application. A sample JDBC solution maintains the state data needed by each servlet as a separate row in an application-maintained JDBC datasource. The primary keys for each row (the data for each servlet) are stored as separate attributes in the `HttpSession`. Using `HttpSession` this way is reduced to the few strings needed to locate the data.

## 6.7.2 Release HttpSession when finished

The application should explicitly and programmatically releases HttpSession. Quite often, programmatic invalidation is part of an application logout function. WebSphere Application Server destroys the allocated HttpSession when it expires (by default, after 1800 seconds or 30 minutes). WebSphere Application Server can only maintain a certain number of HttpSession in memory. When this limit is reached, WebSphere Application Server serializes and swaps the allocated HttpSession to disk. In a high volume system, the cost of serializing many abandoned HttpSession can be quite high.

To avoid this problem, explicitly remove the session by using the following code:

```
javax.servlet.http.HttpSession.invalidate();
```

## 6.7.3 Do not create HttpSession in JSPs by default

By default, JSP files create HttpSession. This is in compliance with J2EE to facilitate the use of JSP implicit objects, which can be referenced in JSP source and tags without explicit declaration. HttpSession is one of those objects. If you do not use HttpSession in your JSP files then you can save some performance overhead. The way to avoid the session being created is to insert the following tag in the JSP code:

```
<%@ page session="false"%>
```

## 6.7.4 Do not use SingleThreadModel

SingleThreadModel is a tag interface that a servlet can implement to transfer its re-entrance problem to the servlet engine. As such, *javax.servlet.SingleThreadModel* is part of the J2EE specification. The WebSphere servlet engine handles the servlet's re-entrance problem by creating separate servlet instances for each user. Because this causes a great amount of system overhead, SingleThreadModel should be avoided.

## 6.7.5 Use the HttpServlet Init() method

Use the `HttpServlet Init()` method to perform expensive operations that only need to be done *once*.

Because the servlet `init()` method is invoked when the servlet instance is loaded, it is the perfect location to carry out expensive operations that only need to be performed during initialization. By definition, the `init()` method is thread-safe. The results of operations in the `HttpServlet.init()` method can be cached safely in servlet instance variables, which become read-only in the servlet service method.

A typical use for this would be to cache any JNDI lookups. Even though WebSphere has had a JNDI cache since version 3.5.2, using your own private cache is still faster. An example of using this technique is shown in Figure 6-3 on page 124.

## 6.7.6 Minimize or eliminate use of System.out.println()

Because it seems harmless, the commonly used application development legacy of using `System.out.println` is overlooked for the performance problem it really is. Because `System.out.println` statements and similar constructs synchronize processing for the duration of disk I/O, they can significantly slow throughput.



Avoid using indiscriminate `System.out.println` statements. State of the art enterprise application development facilities, such as IBM VisualAge Java, provide developers with excellent debugging tools for unit testing. Moreover, the WebSphere Application Server Distributed Debugger can be used to diagnose code on a running system.

However, even with these tools, there remains a legitimate need for application tracing both in test and production environments for error and debugging situations. Such application-level tracing like most system level traces should be configurable to be activated in error and debugging situations only. One good design implementation is to tie tracing to a “final boolean” value, which when configured to *false* will optimize both the check and execution of the tracing at compile time. IBM offers an excellent logging tool through the alphaWorks Web site (see “Referenced Web sites” on page 213) called JLOG, which can help you solve this problem.

Consider also that the WebSphere Application Server product allows for the complete deactivation of `System.out` and `System.err` for any given application server at runtime. This is achieved by setting the file names to "" (through the Administrator Console).

### **6.7.7 Don't use `Beans.instantiate()` to create new bean instances**

The method, `java.beans.Beans.instantiate()`, creates a new bean instance either by retrieving a serialized version of the bean from disk or creating a new bean if the serialized form does not exist. The problem, from a performance perspective, is that each time `java.beans.Beans.instantiate` is called, the file system is checked for a serialized version of the bean. As usual, such disk activity in the critical path of your Web request can be costly. To avoid this overhead, simply use “new” to create the instance.

### **6.7.8 Use and reuse `DataSources` for JDBC connections**

To avoid the overhead of acquiring and closing JDBC connections, WebSphere Application Server provides JDBC connection pooling based on JDBC 2.0. Servlets should use WebSphere Application Server JDBC connection pooling instead of acquiring these connections directly from the JDBC driver. WebSphere Application Server JDBC connection pooling involves the use of `javax.sql.DataSources`.

The application should cache the `DataSource` object for the best performance. Figure 6-3 shows an example of how to do this.

```

public class GoodJDBCServlet extends HttpServlet {

    private javax.sql.DataSource ds = null;    // For Caching the DataSource

    // Get the DataSource (Exception Handling removed for clarity)
    public void init(ServletConfig config) throws ServletException {
        super.init(config);
        java.util.Hashtable env = new java.util.Hashtable();
        env.put(Context.INITIAL_CONTEXT_FACTORY,
            "com.ibm.ejs.ns.jndi.CNInitialContextFactory");
        Context ctx = new InitialContext(env);
        // Store the DataSource for use by every instance
        ds = (javax.sql.DataSource)ctx.lookup("jdbc/SAMPLE");
        ctx.close();
    }

    // Get a pooled connection
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        try { // Get connection and execute Query
            Connection conn = ds.getConnection(USERID,PASSWORD);
            PreparedStatement pStmt = conn.prepareStatement("select * from SCHEMA.SOMETABLE");
            ResultSet rs = pStmt.executeQuery();
            ...
        } finally { // Always close both the preparedStatement and the Connection
            if (pStmt!=null) pStmt.close(); // Note: Wrap this statement in try..catch
            if (conn!=null) conn.close();   // Note: Wrap this statement in try..catch
        }
    }
}

```

*Figure 6-3 Good JDBC coding technique*

A `javax.sql.DataSource` is obtained from WebSphere Application Server through a JNDI naming lookup. Avoid the overhead of acquiring a `javax.sql.DataSource` for each SQL access. This is an expensive operation that will severely impact the performance and scalability of the application. Instead, servlets should acquire the `javax.sql.DataSource` in the `Servlet.init()` method (or some other thread-safe method) and maintain it in a common location for reuse.

### 6.7.9 Release JDBC resources when done

Failing to close and release JDBC connections can cause other users to experience long waits for connections. Although a JDBC connection that is left unclosed will be reaped and returned by WebSphere Application Server after a time-out period, others may have to wait for this to occur. Close JDBC statements when you are finished with them. JDBC ResultSets can be explicitly closed as well. If not explicitly closed, ResultsSets are released when their associated statements are closed. Ensure that your code is structured to close and release JDBC resources in all cases, even in exception and error conditions.

## 6.8 Coding considerations for EJBs

Using EJBs with their own persistence mechanism provides for a separate set of performance problems. This section covers some of the coding issues that relate to EJBs only.

## 6.8.1 Access entity beans from session beans

Avoid accessing EJB entity beans from client or servlet code. Instead wrap and access EJB entity beans in EJB session beans. This satisfies two performance concerns:

- ▶ *Reducing the number of remote method calls:* When the client application accesses the entity bean directly, each getter method is a remote call. A wrapping session bean can access the entity bean locally and collect the data in a structure, which it returns by value.
- ▶ *Providing an outer transaction context for the EJB entity bean:* An entity bean synchronizes its state with its underlying data store at the completion of each transaction. When the client application accesses the entity bean directly, each getter method becomes a complete transaction. A store and a load follow each method. When the session bean wraps the entity bean to provide an outer transaction context, the entity bean synchronizes its state when outer session bean reaches a transaction boundary.

The WebSphere Application Server ships a complete command framework, which formalizes a solution to the above problem. The command facility is implemented in the `com.ibm.websphere.command` Java package.

## 6.8.2 Reuse EJB homes

EJB homes are obtained from WebSphere Application Server through a JNDI naming lookup. This is an expensive operation that can be minimized by caching and reusing EJB Home objects. For simple applications, it might be enough to acquire the EJB home in the servlet `init()` method (similar to the concept shown in Figure 6-3). More complicated applications may require cached EJB homes in many servlets and EJBs. One possibility for these applications is to create an EJB Home Locator and Caching class.

## 6.8.3 Remove stateful session beans when finished

Instances of stateful session beans have affinity to specific clients. They remain in the container until they are explicitly removed by the client, or removed by the container when they time-out. Meanwhile, the container might need to passivate inactive stateful session beans to disk. This requires overhead for the container and constitutes a performance hit to the application. If the passivated session bean is subsequently required by the application, the container activates it by restoring it from disk. By explicitly removing stateful session beans when finished with them, applications will decrease the need for passivation and minimize container overhead. To remove a stateful session bean, simply use:

```
mySessionBean.remove();
```

## 6.9 EJB deployment considerations

In addition to coding considerations, there are also a number of factors in how the deployment descriptor is defined, which can impact performance significantly. This section covers a few of those.

### 6.9.1 Use read-only methods where appropriate

When setting the deployment descriptor for an EJB entity bean, you can mark “getter” methods as “read-only.” If a transaction unit of work includes no methods other than “read-only” designated methods, then the Entity Bean state synchronization will not invoke store. This is a simple technique to reduce the number of redundant commits.

## 6.9.2 Reduce the transaction isolation level where appropriate

By default, most developers deploy EJBs with the transaction isolation level set to TRANSACTION\_SERIALIZABLE. This used to be the default in IBM VisualAge Java - Enterprise Edition and still is in other EJB deployment tools. It is also the most restrictive and protected transaction isolation level incurring the most overhead.

Most workloads do not require the isolation level and protection afforded by TRANSACTION\_SERIALIZABLE. This isolation level has a major performance impact and shouldn't be used unless absolutely necessary. A given application might never update the underlying data or be run with other concurrent updaters.

In that case, the application would not have to be concerned with dirty, non-repeatable, or phantom reads TRANSACTION\_READ\_UNCOMMITTED would probably be sufficient.

Because the EJB transaction isolation level is set in its deployment descriptor, the same EJB could be reused in different applications with different transaction isolation levels. Review your isolation level requirements, and adjust them appropriately to increase performance.

**Note:** The isolation level cannot change during a transaction. Therefore, common practice is to put all beans in the application at the same isolation level. However, setting a few methods at a different isolation level is possible when necessary, if done carefully.



## Case study

This chapter walks through a case study in which we gradually improve the system performance by performing different activities and using several performance analysis tools. It shows every step that we take and all the changes that we make to the system. It also shows how these changes affect our workload throughput on the system.

The purpose of this chapter is to show a practical example of performance methodology, rather than how many improvements you can get with a particular activity. Keep in mind that the results and improvements we achieved in this case study may not apply to every situation because all the run environments are different from each other.

We have a workload that is exactly the same in every run. It consists of placing 1000 orders from 200 clients (Web browsers) simultaneously. In the first run, the system performs so badly that it takes 130 minutes to complete the workload and still only 106 out of 1000 orders are processed correctly. At the end of our tuning activities, we manage to process all 1000 orders and reduce the runtime to only 4 minutes and 36 seconds.

To assure consistency in measurements, we minimize the network effect by putting the clients (Web browsers) on one workstation that is directly attached to the iSeries server via a private LAN. We use an IBM internal tool AKstress to simulate 200 Web browsers issuing requests simultaneously. Although we access the same communication IOP, the communication IOP is not a bottleneck.

Our approach is to:

1. Tune up the OS/400.
2. Verify the HTTP server and WebSphere Application Server settings.
3. Inspect the application itself.

As it happens in real life, we repeat these steps in sequence several times before we are satisfied with the results.

## 7.1 System used in this case study

The iSeries server we use in this case study has following configuration:

### ► Hardware

- iSeries server Model 270, processor feature #2434
- Two SStar 600 Mhz central processors with 2350 CPW total; no interactive CPW
- 4 GB of main storage with 4 MB L2 cache
- 12 RAID protected disks with total capacity of 175GB of which 17 percent was in use

### ► Software

- OS/400 V5R1
- WebSphere Application Server Version 3.5.4: At the end, we compare it with performance of WebSphere Application Server Version 4.0
- IBM HTTP server (Original): We make one run with IBM HTTP Server (powered by Apache) and compare both results.

## 7.2 Application used in this case study

This section describes the OrderEntry application example. This application is representative of a commercial application, although it does not include all of the necessary error handling that a business application requires. We use this application to demonstrate various performance measurement, analysis, and tuning techniques for iSeries WebSphere-based applications.

### 7.2.1 Overview of the OrderEntry application

The ABC Company is a wholesale supplier with one warehouse and 10 sales districts. Each district serves 3000 customers (30000 total customers for the company). The warehouse maintains stock for the 100000 items sold by the Company.

Figure 7-1 illustrates the company structure (warehouse, district, and customer).

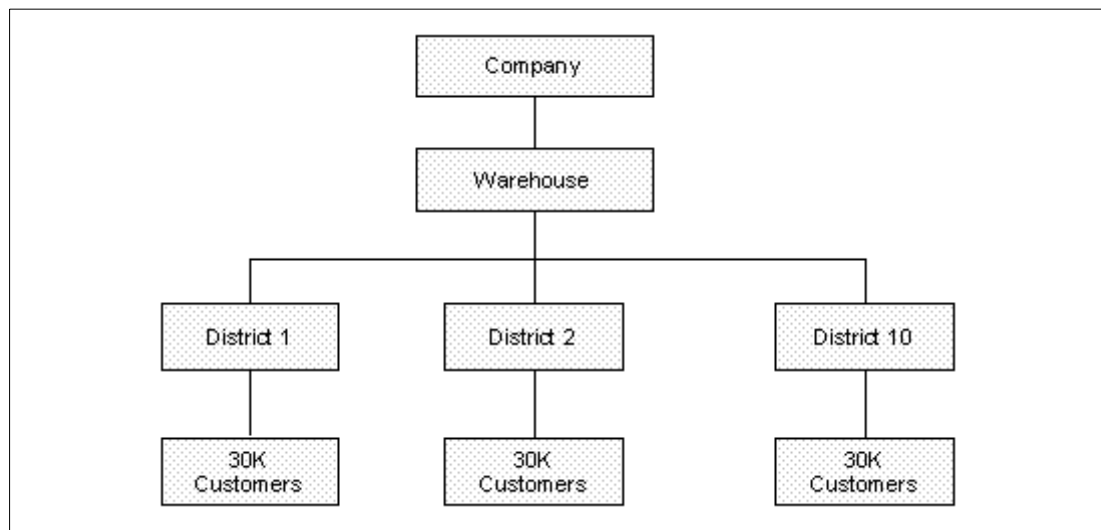


Figure 7-1 The company structure

The company runs its business with a database. This database is used in a mission critical, online transaction processing (OLTP) environment. The database includes tables with the following data:

- ▶ District information (next available order number, tax rate, and so on)
  - ▶ Customer information (name, address, telephone number, and so on)
  - ▶ Order information (date, time, shipper, and so on)
  - ▶ Order line information (quantity, delivery date, and so on)
  - ▶ Item information (name, price, item ID, and so on)
  - ▶ Stock information (quantity in stock, warehouse ID, and so on)
- OrderEntry application architecture with objects

The ABC Company order entry application is a Java-based application. It runs on the iSeries server in the WebSphere Application Server environment. It is Enterprise JavaBeans-based and provides both a Java client-based interface and a browser-based interface.

## 7.2.2 Application architecture

The application architecture uses the existing database tables and builds an object architecture around them. When building an object based architecture, it is important to name the identifiable objects in the application. A variety of methods exist for creating an object-oriented application. One common way is to describe the process and highlight the nouns, as a starting point for identifying the objects.

In our example, a *customer* resides in a particular *district*. The customer contacts an order entry clerk to place an *order* for one or more *items*. Each item appears as an *order detail line* in the order. Before the order can be placed, the clerk checks to see if there is enough of the item in *stock*.

This list of objects corresponds to the tables that also identify the fields contained in the tables. An object diagram corresponding to these tables is shown in Figure 7-2.



Figure 7-2 Application object diagram

In Figure 7-2, we make one further refinement. We separate out an address object to hold the common address information that is required by the customer and the district.

This list of objects represents all of the tables. However, the description includes another noun – the order entry clerk (OrderEntryClerk). This object becomes important in the next step of identifying objects – identifying the relationships between the objects. The OrderEntryClerk is an object with the primary purpose of acting on other objects. The relationships between the identified objects are either containment or “uses” relationships. Figure 7-3 shows all of the objects that were previously identified and the relationships between them.



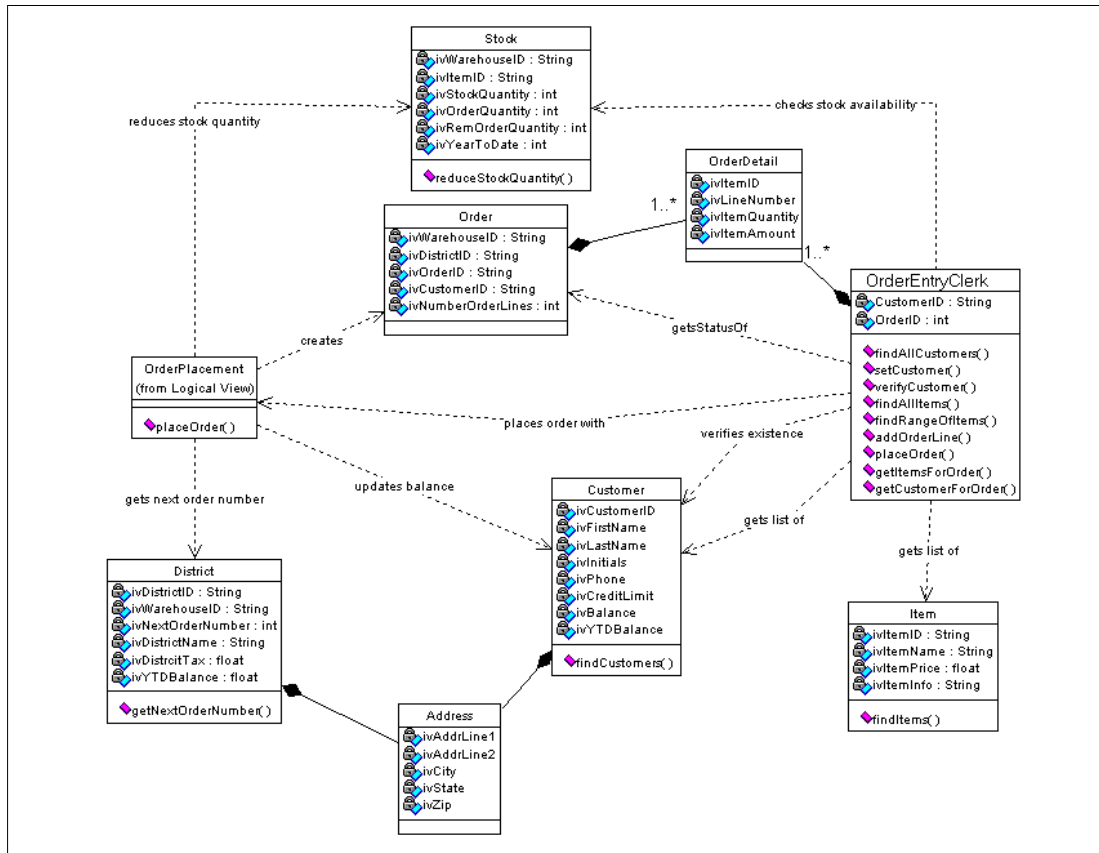


Figure 7-3 Object relationships

Both the customer and the district class contain an address. This provides reuse, one of the key tenets of object-oriented programming. As shown later in this chapter, as a goal, reuse may need to be overlooked to gain other advantages, such as making use of some of the programming facilities. The order contains a collection of order details. Because the order entry clerk initially takes down the order information, it also needs to contain a collection of order details (OrderDetails). This collection of order details is used as a parameter when calling the method necessary to create a new order.

The “uses” relationships primarily involve the OrderEntryClerk, as well. The OrderEntryClerk acts on several different objects. It retrieves a list of customers when the order is initiated. Later, when an order is being placed, it verifies the existence of a specific customer. It also retrieves a list of items to be ordered. As requests for creating order details are received by the OrderEntryClerk, it checks the stock availability of the item being ordered. After all of the order details are created, the order is placed through the OrderEntryClerk.

To this point, all of the actions that have been described reflect only as reads of the database tables. Placing the order is the activity that involves writing to the tables represented by the objects. Because this is a complex task and represents a business transaction, OrderPlacement is represented as a separate object. The OrderPlacement object also acts on several different objects. It creates a new order object. Because it receives a collection of order details as a parameter, it merely passes them on to the order for them to be created. The OrderPlacement object also retrieves the next order number from the district, updates the customer balances, and reduces the stock quantity. All of these updates are completed as a single unit of activity and, therefore, are encapsulated in a transaction, as shown later in this chapter.

The objects described in this section serve one of two purposes. They either represent data that is maintained in a table, or they represent actions on the data or business tasks. These two types of objects correspond to entity and session beans, respectively.

### Business data: Entity Enterprise JavaBeans

Entity beans are persistent objects that represent data stored in some persistent fashion, such as a database or an existing legacy application. When you modify the variables for the bean, you modify the data in the database. In our case, the persistent store is the relational database on the iSeries server. Our analysis identified the following five entity beans:

- ▶ Stock
- ▶ Item
- ▶ Customer
- ▶ Order
- ▶ District

### Business processes: Session Enterprise JavaBeans

Session beans are non-persistent objects that run on the server and implement business logic. Our example uses two session beans:

- ▶ OrderEntryClerk
- ▶ OrderPlacement

### Three-tier versus two-tier architecture

Session beans allow you to easily implement a three-tier architecture. The traditional client/server architecture is a two-tiered architecture in which there are many clients and a single database. The clients often implement the business logic as well as the user interface. In a three-tiered architecture, business logic is moved out of the many clients and into a session bean running in an EJB server. The clients talk to the session bean and the bean talks to the databases.

This architecture makes the client code smaller and easier to maintain. You can change the business logic without redistributing any client code. More importantly, this architecture lets you share business logic across different types of clients. If you implement your business logic in a bean, you can use that same bean in a Java application, an applet, a servlet or a non-Java application as shown in the Figure 7-4.

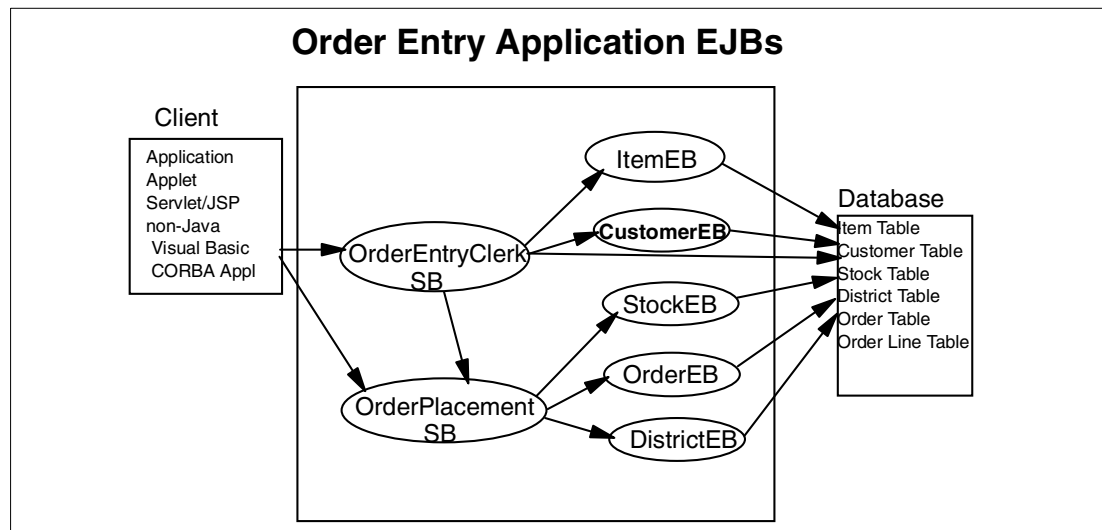


Figure 7-4 Three-tier architecture

### ***Stateless or stateful***

A *stateless session bean* allows you to write code that runs on the server and can be reused by many clients. However, the stateless session bean does not remember anything about the client between method invocations. In fact, if your client calls two methods of a stateless session bean, there is no guarantee that both methods are called on the same object in the server.

Conversely, a *stateful session bean* remembers information between method calls. To use a trivial example, a stateful session bean can have two methods: `setName` and `getName`. A client can pass a name to the bean using `setName` and retrieve that name in another call using `getName`.

In our example application, the *OrderPlacement bean* is a stateless session bean and the *OrderEntryClerk bean* is a stateful session bean. The *OrderPlacement* bean encapsulates the business logic to place an order. The *OrderEntryClerk* encapsulates the ordering process. The *OrderEntryClerk* bean is used by GUI programs such as Java applications or applets. The *OrderPlacement* bean is used by the *OrderEntryClerk* bean to actually place the order.

## **7.2.3 A customer transaction**

A customer transaction occurs based on the following series of events:

1. Customers telephone one of the 10 district centers to place an order.
2. The district customer service representative answers the telephone, obtains the following information, and enters it into the application:
  - Customer number
  - Item numbers that the customer wants to order
  - The quantity required for each item
3. The customer service representative may prompt for a list of customers or a list of parts.
4. The application then performs the following actions:
  - a. Reads the customer last name, customer discount rate, and customer credit status from the Customer Table (CSTMR).
  - b. Reads the District Table for the next available district order number. The next available district order number increases by one and is updated.
  - c. Reads the item names, item prices, and item data for each item ordered by the customer from the Item Table (ITEM).
  - d. Checks if the quantity of ordered items is in stock by reading the quantity in the Stock Table (STOCK).
5. When the order is accepted, the following actions occur:
  - a. Inserts a new row into the Order Table to reflect the creation of the new order (ORDERS).
  - b. A new row is inserted into the Order Line Table to reflect each item in the order.
  - c. The quantity is reduced by the quantity ordered.
  - d. The customer record is updated to reflect year to date purchases.
  - e. A message is written to a data queue to initiate order printing.

## 7.2.4 Client application design

To make it easy for clients to use our Enterprise JavaBeans, we provide the `ItemsDb` class. It provides a level of abstraction for client applications. Our goal is to allow client applications to use our Enterprise JavaBeans without having to deal with the implementation details and complexity. The following key methods are provided:

- ▶ The `getInitialContext` method creates an `InitialContext` object.
- ▶ The `connect` method establishes a connection to the Java server.
- ▶ The `getAllCustomers` method retrieves a list of customers from the server.
- ▶ The `getItems` method retrieves a list of items from the server.
- ▶ The `findRangeOfItems` method retrieves a subset of items from the server.
- ▶ The `verifyCustomer` method verifies that a customer number is valid.
- ▶ The `confirmOrder` method places and confirms an order.
- ▶ The `submitOrder` method places and confirms an order using a shopping cart.
- ▶ The `connectStateless` method establishes a stateless connection to the Java server for servlets.
- ▶ The `submitOrderStateless` method places an order given a customer ID and a shopping cart.

All the methods in the `ItemsDb` class, except `connectStateless` and `submitOrderStateless`, use the `OrderEntryClerk` session Enterprise JavaBean to access iSeries resources. The "stateless" methods use the `OrderPlacement` stateless session EJB.

Our intention is to create an access class that is capable of running on its own. That is, you can use it even without any user interfaces. As shown in Figure 7-5, the advantage of this approach is that you can use the access class in any context. For example, you can use a graphical user interface (GUI) on top of the access class when you want to create a stand-alone Java application. You also use exactly the same class when you want to develop a Java servlet. In this case, replace the GUI front end with a layer that is capable of running as a servlet.

Separating the layers properly and using a clean and well-defined interface between the layers is important if you want to take advantage of object-orientation.

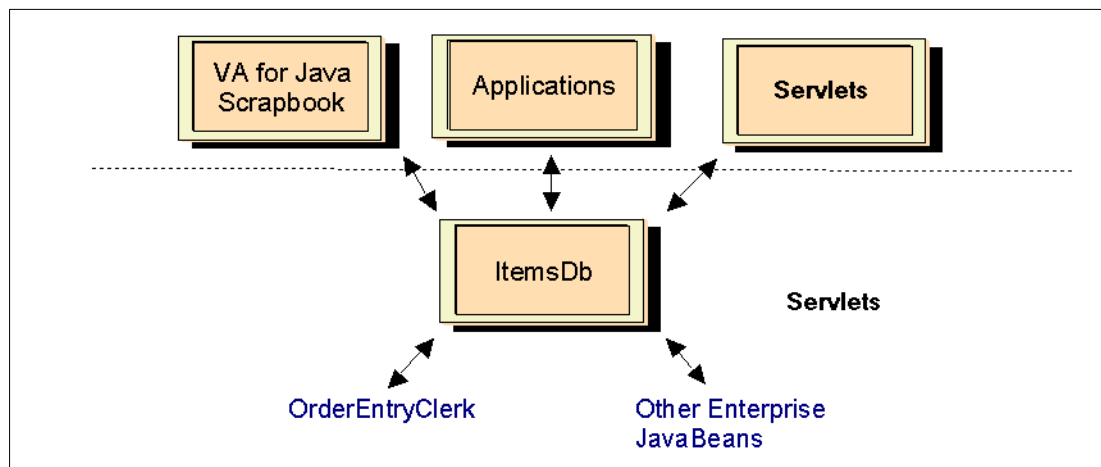


Figure 7-5 Interfacing to the `ItemsDb` class

### The Java application client

A Java application is used to interface with the EJB application. The Java client application provides a graphical user interface for placing orders. It can be used by an order entry clerk for handling customer requests.

Figure 7-6 shows the Java application graphical user interface. The Java application does not directly interface with the Enterprise JavaBeans. It uses the abstraction layer, the ItemsDb class, which makes it easier to connect to the EJBs.

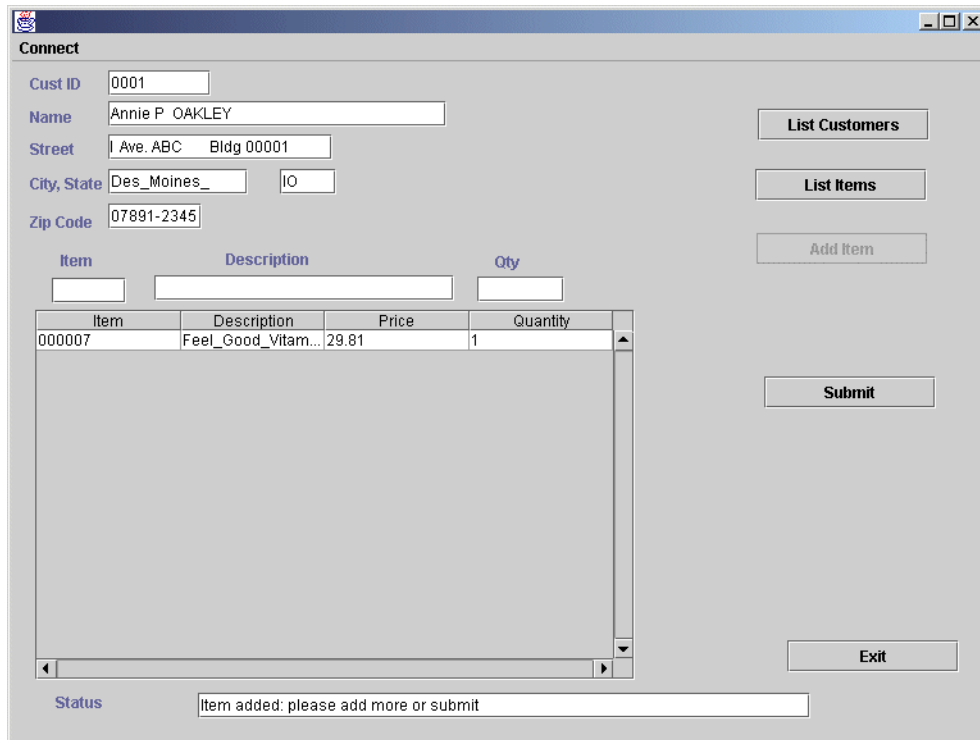


Figure 7-6 Running the Java client

As shown in Figure 7-7, the ItemsDb class contains all the code necessary for interacting with the EJB back end. The Java application uses the ItemsDb class to interface with the OrderEntryClerk EJB. In turn, it interfaces with the OrderPlacement EJB.

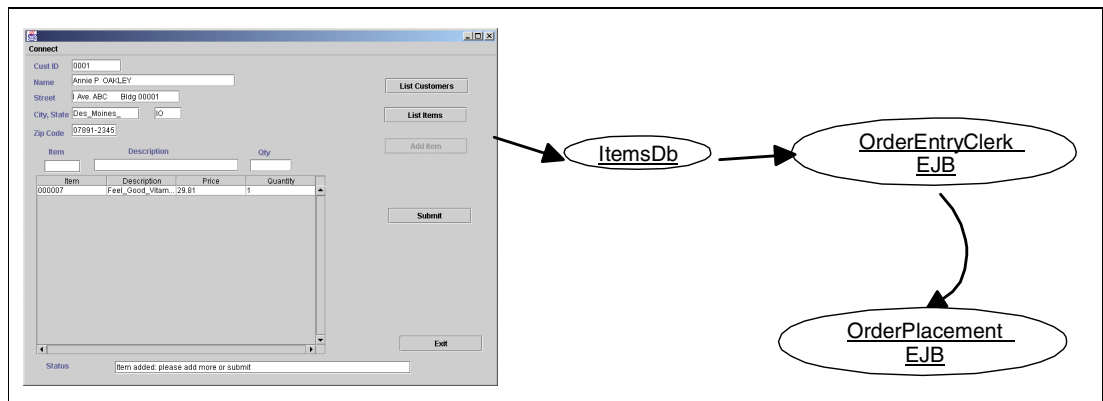


Figure 7-7 The OrderEntry application

## The servlet client

A servlet-based Web application is also provided. This allows the OrderEntry application to be accessed from a Web browser.

Servlets are inherently multithreaded. Stateful session beans are used to maintain the state for one client. Even if multiple Web users may be accessing the servlet, multithreading makes it appear as if the servlet is one client. State is not automatically maintained across method calls for each Web client. This means that if we're going to use a stateful session bean, we're going to have to explicitly manage that connection.

The easiest and usually best way to connect servlets to session beans is to manage state information within the servlet, using stateless beans to perform business tasks. The *stateless* methods in our ItemsDb class will do this.

As shown in Figure 7-8, we use two servlets. They are named ItemSessionServlet and CartServlet.

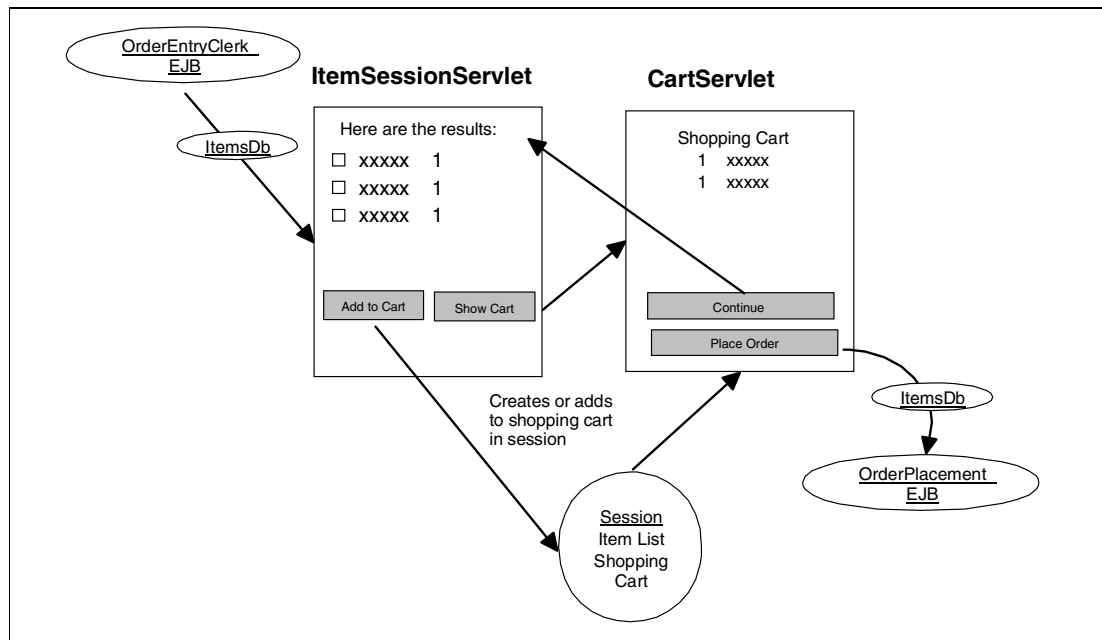


Figure 7-8 Order Entry servlets

### ItemSessionServlet

The ItemSessionServlet servlet is fairly simple. It displays a list of the items available for ordering. It also allows the user to select items to add to their shopping cart. The connection occurs in exactly the same way as we saw previously in the application using the ItemsDb class. It uses the ItemsDb class to interface to the OrderEntryClerk EJB to display the list of items.

### CartServlet

The CartServlet servlet is a command-driven servlet that supports adding items to a shopping cart and placing an order. It uses the ItemsDb class to interface with the OrderPlacement EJB to actually place the order.

### Servlet application flow

The application flow happens like this:

1. When the ItemSessionServlet is started, it connects to the OrderEntryClerk session bean using the ItemsDb class. It maintains this object, so all threads use the same object.
2. When a user executes the ItemSessionServlet, it causes the doPost() method to execute. The doPost() method executes the findRangeOfItems() method of the ItemsDb class. It

returns a Vector of items, which is used to populate the HTML table displayed on the form. This is shown in Figure 7-9.

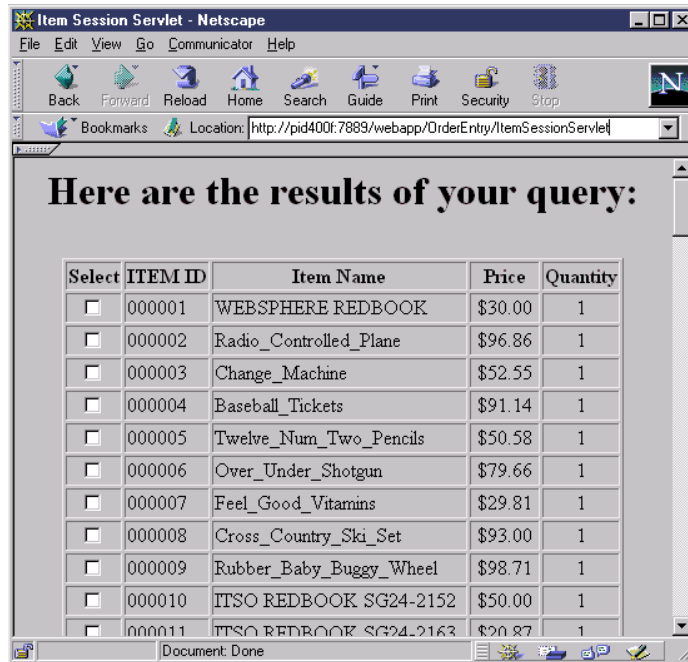


Figure 7-9 ItemSession servlet

3. The user selects the items they want to order and clicks the *Add to Cart button* on the form, shown in Figure 7-10.

<input type="checkbox"/>	000092	Old_Wooden_Toothpicks	\$4.00	1
<input type="checkbox"/>	000093	8_Foot_Plant_Pole	\$97.14	1
<input type="checkbox"/>	000094	Zoo_Season_Pass	\$290.88	1
<input type="checkbox"/>	000095	Lone_Ranger_Costume	\$14.67	1
<input type="checkbox"/>	000096	Compact_Disc_Player	\$68.67	1
<input type="checkbox"/>	000097	Bear_Claw_Doughnuts	\$0.50	1
<input checked="" type="checkbox"/>	000098	Four_Drawer_Cabinets	\$69.62	1
<input checked="" type="checkbox"/>	000099	Orange_Golf_Balls	\$61.70	1
<input type="checkbox"/>	000100	Street_Hockey_Balls	\$13.69	1

Figure 7-10 Add to Cart button

4. This posts the form to the CartServlet using the Add to Cart command. The form shown in Figure 7-11 appears. The CartServlet creates a shopping cart object in the servlet session object.

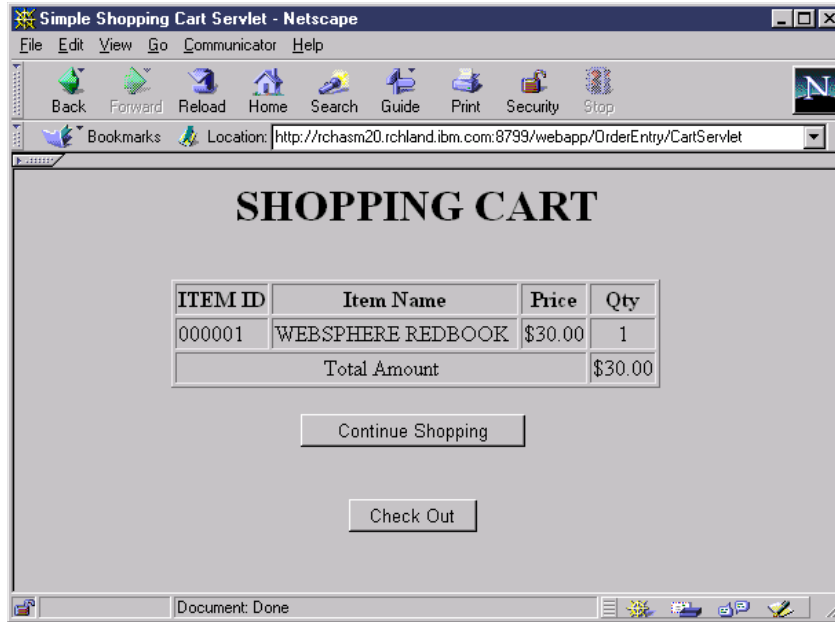


Figure 7-11 The servlet shopping cart

5. When the user is done shopping, they click the *Check Out* button on the form to place an order. This action passes the Check Out command to the CartServlet servlet.
6. The CartServlet servlet posts a verification form, shown in Figure 7-12, and requests a customer number.

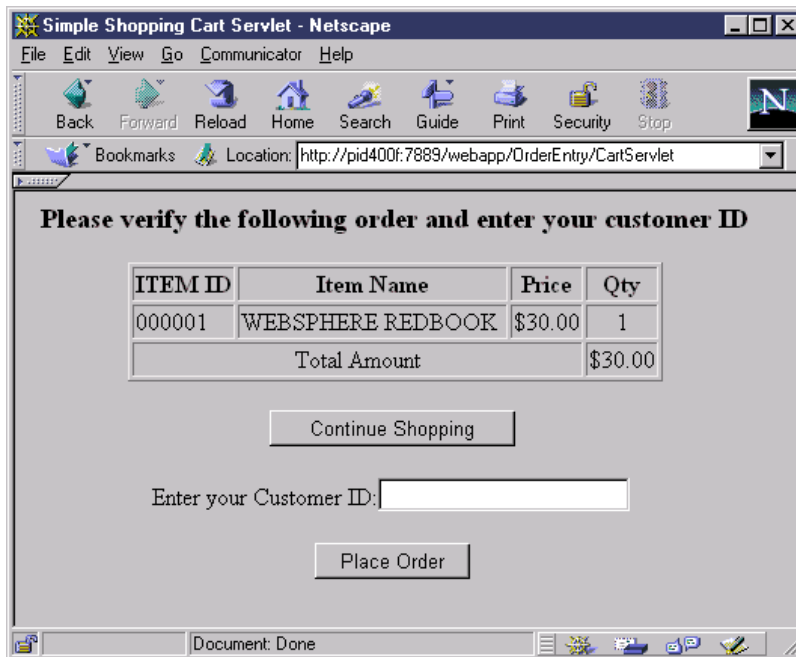


Figure 7-12 CartServlet servlet

7. The user enters a customer number and clicks the *Place Order* button. This causes the placeOrder method to be called. It calls the connectStateless and submitOrderStateless methods in the ItemsDb class, which in turn connect to the OrderPlacement session bean (which is stateless) and submits the order.



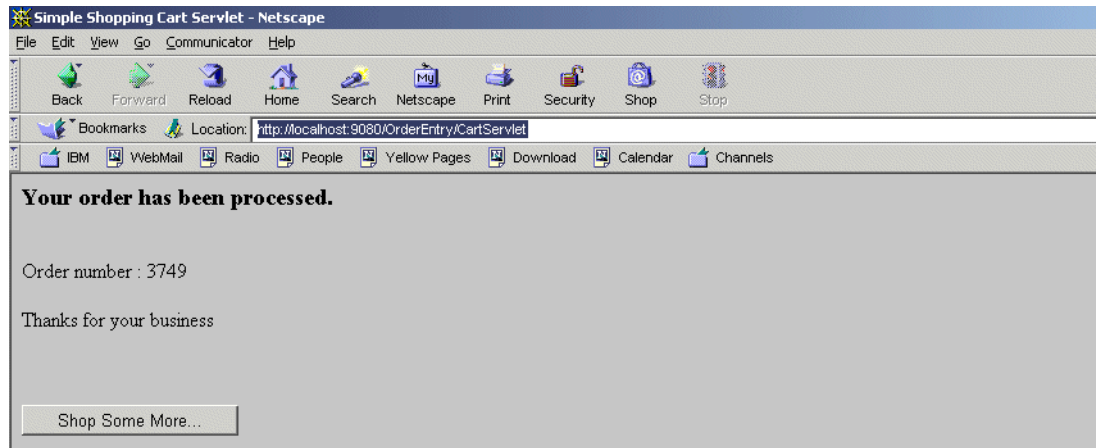


Figure 7-13 A successful order

8. If the order processing is successful, the order number is returned to the CartServlet and displayed on the browser.

## 7.2.5 OrderEntry application database layout

The sample application uses the following tables of the database. Each table is described in the following section, indicating the layout of the database.

- ▶ District
- ▶ Customer
- ▶ Order
- ▶ Order line
- ▶ Stock
- ▶ Item (catalog)

### Tables

Table 7-1 District Table Layout (DSTRCT)

Field name	Real name	Type	Length
DID	District ID	Decimal	3
DWID	Warehouse ID	Character	4
DNAME	District Name	Character	10
DADDR1	Address Line 1	Character	20
DADDR2	Address Line 2	Character	20
DCITY	City	Character	20
DSTATE	State	Character	2
DZIP	Zip Code	Character	10
DTAX	Tax	Decimal	5
DYTD	Year to Date Balance	Decimal	13
DNXTOR	Next Order Number	Decimal	9
Primary Key: DID and DWID			

Table 7-2 Customer Table Layout (CSTMR)

Field name	Real name	Type	Length
CID	Customer ID	Character	4
CDID	District ID	Decimal	3
CWID	Warehouse ID	Character	4
CFIRST	First Name	Character	16
CINIT	Middle Initials	Character	2
CLAST	Last Name	Character	16
CADDR1	Address Line 1	Character	20
CCREDIT	Credit Status	Character	2
CADDR2	Address Line 2	Character	20
CDCT	Discount	Decimal	5
CCITY	City	Character	20
CSTATE	State	Character	2
CZIP	Zip Code	Character	10
CPHONE	Phone Number	Character	16
CBAL	Balance	Decimal	7
CCRDLM	Credit Limit	Decimal	7
CYTD	Year to Date	Decimal	13
CPAYCNT	Payment	Decimal	5
CDELCNT	Delivery Qty	Decimal	5
CLTIME	Time of Last Order	Numeric	6
CDATA	Customer Information	Character	500
Primary Key: CID, CDID, and CWID			

Table 7-3 Order Table Layout (ORDERS)

Field name	Real name	Type	Length
OWID	Warehouse ID	Character	4
ODID	District ID	Decimal	3
OCID	Customer ID	Character	4
OID	Order ID	Decimal	9
OENTDT	Order Date	Numeric	8
OENTTM	Order Time	Numeric	6
OCARID	Carrier Number	Character	2

Field name	Real name	Type	Length
OLINES	Number of Order Lines	Decimal	3
OLOCAL	Local	Decimal	1
Primary Key: OWID, ODID, and OID			

Table 7-4 Order Line Table Layout (ORDLIN)

Field name	Real name	Type	Length
OID	Order ID	Decimal	9
ODID	District ID	Decimal	3
OWID	Warehouse ID	Character	4
OLNBR	Order Line Number	Decimal	3
OLSPWH	Supply Warehouse	Character	4
OLIID	Item ID	Character	6
OLQTY	Quantity Ordered	Numeric	3
OLAMNT	Amount	Numeric	7
OLDLVD	Delivery Date	Numeric	6
OLDSTI	District Information	Character	24
Primary Key: OLWID, OLDID, OLOID, and OLNBR			

Table 7-5 Item Table Layout (ITEM)

Field name	Real name	Type	Length
IID	Item ID	Character	6
INAME	Item Name	Character	24
IPRICE	Price	Decimal	5
IDATA	Item Information	Character	50
Primary Key: IID			

Table 7-6 Stock Table Layout (STOCK)

Field name	Real name	Type	Length
STWID	Warehouse ID	Character	4
STIID	Item ID	Character	6
STQTY	Quantity in Stock	Decimal	5
STDIO1	District Information	Character	24
STDIO2	District Information	Character	24

Field name	Real name	Type	Length
STDI03	District Information	Character	24
STDI04	District Information	Character	24
STDI05	District Information	Character	24
STDI06	District Information	Character	24
STDI07	District Information	Character	24
STDI08	District Information	Character	24
STDI09	District Information	Character	24
STDI10	District Information	Character	24
STYTD	Year to Date	Decimal	9
STORDERS	Number of orders	Decimal	5
STREMORD	Number of remote orders	Decimal	5
STDATA	Item Information	Character	50
Primary Key: STWID and STIID			

## 7.3 Tuning the OS/400

The problem we want to solve in this case study is simple. The iSeries server is running the OrderEntry application with 99.9% CPU utilization. It takes a long time (130 minutes) to run 1000 orders and still the transaction throughput was bad. Only 106 out of 1000 orders are processed correctly.

We perform the following steps:

1. Set the system values as described in 4.3.2, “Verifying the settings of system values” on page 61.
2. Check the PTF levels as described in 4.2, “Verifying the state of Licensed Program Products” on page 58.
3. Use Management Central’s System Monitor to observe system usage.
4. Use the WRKSYSSTS command to observe the memory usage as described in “Tuning main storage with the WRKSYSSTS command” on page 72.
5. Start examining the subsystem descriptions as shown in Figure 7-14. This display shows a typical Management Central System Monitor window that we observed while running our workload.

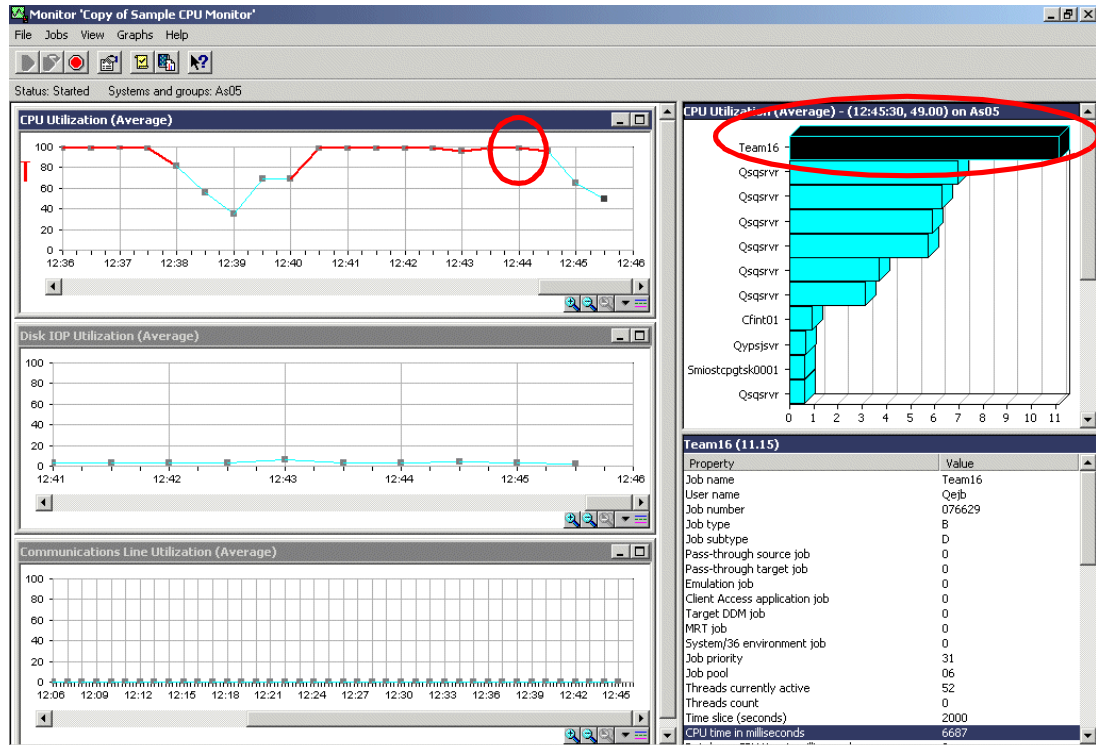


Figure 7-14 System Monitor

We set the System Monitor to show us average CPU, disk IOP, and communications IOP utilization. In our case, the CPU utilization is 100% during this run (top left panel in Figure 7-14). By clicking the CPU utilization graph (circled spot in the top left panel), you see CPU utilization *by job* in a selected time interval, shown in the top right panel. By clicking the bar next to the job (arrow in top right panel), you see the job details shown in the bottom right panel. You see that Disk IOP (middle left panel) and Communication IOP (bottom left panel) are not a bottleneck since utilization for both of these is in the single digits all the time.

### 7.3.1 Examining and tuning the QSYSWRK subsystem

QSQRVR prestart jobs that are used to run the database access for our application run in subsystem QSYSWRK. We found out that the QSQRVR jobs were directed to a separate memory SHRPOOL1. This is a good idea when there is a legacy (conventional, green screen) workload running on a system because it ensures that the workloads are not contending the same resources. In this case, however, the work was not done all the way through. The workload was completed in 130 minutes, 5 seconds, but 994 of the 1000 transactions were rolled back.

The activity level for the memory pool was way too small. Memory pool \*SHRPOOL1 had 600 MB of main storage and only five activity levels available in the pool.

When observing the WRKACTJOB display while the workload was running, we found that there were between seven and ten active QSQRVR jobs having a status of RUN simultaneously (see “Using the search facility on the WRKACTJOB display” on page 33 for details). Having only five activity levels available for these jobs, this meant that there were two to five QSQRVR jobs in ineligible status at any given time. If a server job, such as QSQRVR does not get the activity level (which, in this case, means get the CPU time) when it is needed, the result is a time out.

We monitor the QSQSRVR jobs' behavior in the QSYSWRK subsystem by performing the following steps:

1. Enter the following command to monitor the QSQSRVR jobs only:  
WRKACTJOB JOB(QSQSRVR)
2. Move the cursor to the Status column and press F16 to rearrange the display by status in alphabetical order.
3. Scroll down until the jobs with RUN status are visible.
4. Press F5 to refresh the screen periodically to see how many jobs are actually running. If the jobs go to *Incl* status, we increase the activity level in the pool using WRKSYSSTS command (see "Tuning main storage with the WRKSYSSTS command" on page 72 for detailed information).

### The effect of changing the activity level setting

At this stage, we decide to increase the number of activity levels that are reserved for the QSQSRVR jobs only. We entered the WRKSYSSTS command. On the display, we gradually increase the activity level setting on \*SHRPOOL1 and run the workload several more times.

These changes bring dramatic improvements. With the activity level finally set to 50, the time for completing the workload goes down to 41 minutes, 37 seconds, where all 1000 orders are successfully completed.

## 7.3.2 Examining and tuning the QEJBSBS subsystem

The jobs related to WebSphere Application Server Version 3.5 run within the QEJBSBS subsystem that resides in library QEJB. We recommend that you create a separate memory pool for WebSphere Application Server instances to ensure that they always have both the memory and the necessary activity levels. We put the QEJBSBS subsystem into memory pool SHRPOOL2.

We measured the memory requirements of the QEJBSBS subsystem by experimenting with various settings for memory pool sizes and activity levels for the pool in which the QEJBSBS subsystem runs. Table 7-7 shows the performance improvements we gained by changing the memory pool sizes and activity level settings.

Table 7-7 The effect of memory and activity level settings to starting WebSphere Application Server

Memory pool size/activity level	Time to start the subsystem (run STRSBS command)	Time to start WebSphere instance (run STRWASINST command)	Time to start WebSphere Administrative Console
150 / 25	80 seconds	110 seconds	400 seconds
250 / 25	43 seconds	88 seconds	165 seconds
500 / 25	42 seconds	40 seconds	101 seconds
500 / 50	49 seconds	28 seconds	96 seconds
500 / 100	45 seconds	28 seconds	90 seconds
500 / 200	41 seconds	26 seconds	50 seconds
500 / 400	38 seconds	30 seconds	49 seconds
1000 / 400	30 seconds	30 seconds	48 seconds

Note that while it was possible to start QEJBSBS subsystem with a pool of 150 MB and 25 activity levels, we could not perform any application transactions with these settings.

**Note:** Based on observations during this test, we recommend a storage pool of 450 MB and activity level of 250 for one Web Application Server. Remember, this may vary in different customer situations.

We monitor the jobs' behavior in the QEJBSBS subsystem by performing the following steps:

1. Enter the following command to monitor the jobs:  
`WRKACTJOB SBS(QEJB/QEJBSBS)`
2. Move the cursor to the Status column and press F16 to rearrange the display in alphabetical order by status.
3. Scroll down to see the jobs with RUN status.
4. Press F5 to refresh the screen periodically to see how many jobs are actually running. If jobs go to *Incl* status, increase the activity level in the pool via the WRKSYSSTS command.

This activity gradually decreased the time needed to run our workload further down to 36 minutes, 23 seconds, with all 1000 orders completed.

### 7.3.3 Summary of system tuning

By performing the system tuning activities, we dramatically decreased our run time from 130 minutes down to 36 minutes and, at the same time, increased the number of processed orders from 106 to 1000. The major contributor was setting memory pool sizes and activity levels. The effect of activity levels and pool sizes on our application throughput and response time is shown in Table 7-8.

Table 7-8 Performance improvements resulted from various pool sizes and activity levels

Settings used/changed	Orders generated	Execution time	Maximum CPU utilization
SHRPOOL1 size 600 / 5	106	130 min. 5 sec.	8%
SHRPOOL1 size 600 / 10	1000	43 min. 22 sec.	100%
SHRPOOL1 size 600 / 50	1000	41 min. 37 sec.	100%
SHRPOOL2 size 600 / 50	1000	41 min.10 sec.	100%
SHRPOOL2 size 600 / 100	1000	39 min. 27 sec.	100%
SHRPOOL2 size 600 / 300	1000	39 min.14 sec.	100%
SHRPOOL2 size 800 / 300	1000	36 min. 41 sec.	100%
SHRPOOL2 size 1800 / 300	1000	36 min. 23 sec.	100%

## 7.4 Tuning WebSphere Application Server

After tuning the system, we tune the WebSphere Application Server. When we performed the activities described in Chapter 5, "Tuning HTTP server and WebSphere Application Server" on page 83, we managed to reduce the runtime for our workload from 36 minutes, 23 seconds to 31 minute and 9 seconds.

While changing the HTTP server and WebSphere Application Server settings, we observed the performance effects using the following tools:

- ▶ Management Central's System Monitor and Job Monitor to measure:
  - System CPU utilization used by WebSphere and application jobs
  - Disk IOP and communication IOP utilization used by WebSphere and application jobs
  - Which jobs consume most of the CPU time
- ▶ WebSphere Resource Analyzer to find out how many database connections and servlets are running concurrently
- ▶ HTTP Server Monitor to observe how many HTTP server threads are active and how many threads are waiting to be processed

First we start Management Central's System Monitor and Job Monitor and have them active while we run our workload. They help us understand what is happening in the system in real-time mode.

The following WebSphere Application Server tuning activities improved the performance in our case study:

- ▶ Running the application in Direct Execution mode instead of JIT
- ▶ Increasing the garbage collection initial size
- ▶ Disabling servlets auto reload
- ▶ Increasing the WebSphere configuration update interval

The results we received after each tuning activity are shown in Table 7-9. The activities were performed in the sequence as shown in this table.

*Table 7-9 Performance improvements of tuning WebSphere Application Servers*

Activity sequence number	Settings used/changed	Orders generated	Execution time	Maximum CPU utilization
1	Running servlets in direct execution mode	1000	35 min. 0 sec.	100%
2	Increasing maximum active threads in HTTP server and servlet engine.	1000	degraded	100%
3	Garbage collection initial size increased to 64 MB	1000	34 min. 5 sec.	100%
4	Garbage collection initial size increased to 128 MB	1000	33 min. 24 sec.	100%
5	Garbage collection initial size increased to 256 MB	1000	32 min. 20 sec.	100%
6	Garbage collection initial size increased to more than 256 MB	1000	same	100%
7	Disable auto reload	1000	31 min. 38 sec.	100%
8	Increasing the WebSphere configuration update interval to 300	1000	31 min. 10 sec.	100%
9	Using Apache Web server	1000	31 min. 9 sec.	100%



## 7.4.1 Direct Execution versus JIT

First, we achieved some improvement by running the application in direct execution mode as opposed to JIT, which was our default. We did this by performing the activities described in 5.2, “Direct execution (DE) versus JIT mode” on page 91.

## 7.4.2 Tuning WebSphere Application Server queues

Next, we try to increase the maximum number of active threads in the queues (HTTP server, servlet engine and database connection). This did not improve the performance in our case study. However, it might help in your particular environment. This section shows you the methodology that we used to find out the optimum numbers or maximum active threads in the HTTP server, servlet engine, and database connection.

We use following tools in this undertaking:

- ▶ WebSphere Resource Analyzer to determine the servlet engine and database connection queues
- ▶ HTTP monitor to determine the optimum size for maximum active HTTP server threads

The default values that we start with are:

- ▶ Maximum HTTP server threads: 40
- ▶ Maximum servlet engine threads: 25
- ▶ Maximum database connections: 10

As you can see, these numbers follow the “funneling” approach where there are more active servlet threads than database connections and more HTTP threads then servlet threads.

Our application runs 200 concurrent clients. We determine the current number of active threads by performing following steps:

1. We start WebSphere Resource Analyzer, expand the node, expand the application server, expand **database connection pools**, expand **servlet engine**, expand **default\_host**, and expand **OrderEntry** application.
2. We click to select the counter. Then we right-click and select **Run** to activate following counters:
  - NativeDS: A datasource that is used by our OrderEntry application
  - OrderEntry application
  - Servlets ItemSessionServlet and CartServlet that are part of the OrderEntry application
3. We run the application and observe the above three counters.

We click ItemSessionServlet to see how many of these servlets run concurrently. As shown in Figure 7-15, we notice that only 24 concurrent ItemSessionServlet servlets are running. Since we run 200 concurrent clients, this tells us that there may be many servlets waiting in the servlet engine queue.

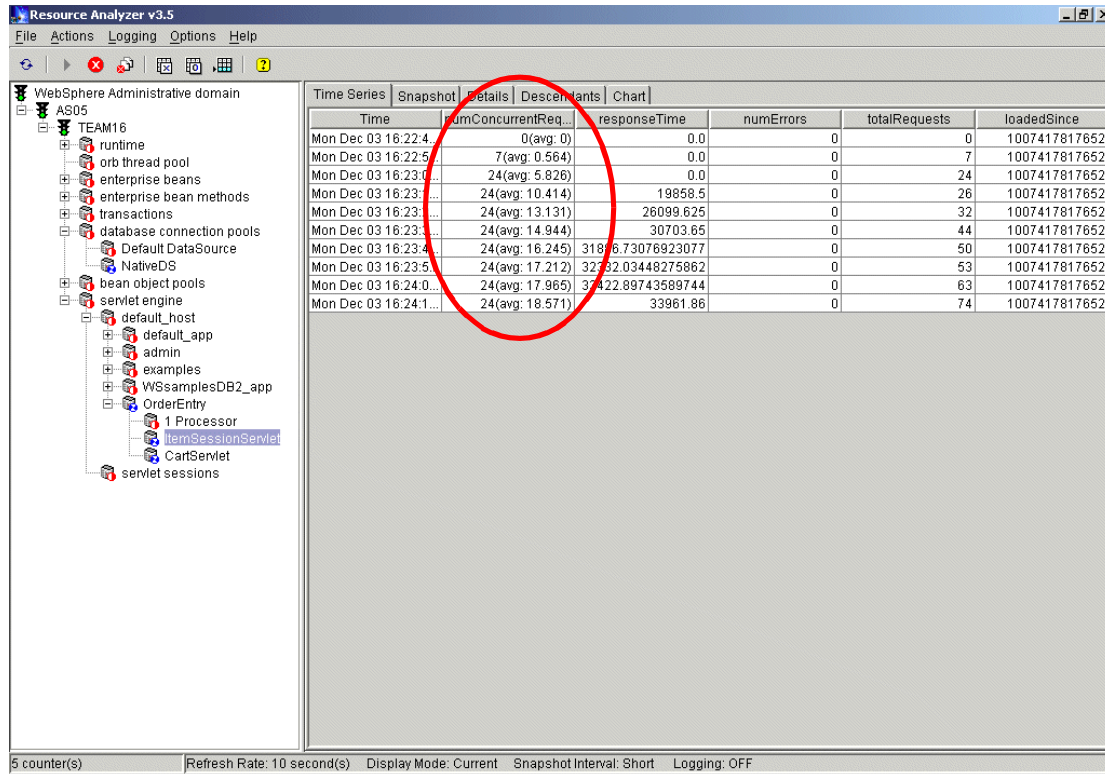


Figure 7-15 Number of concurrent servlets

4. We click **NativeDS** (the data source used by OrderEntry application) to observe the number of concurrent database connections in the connection pool (Figure 7-16).

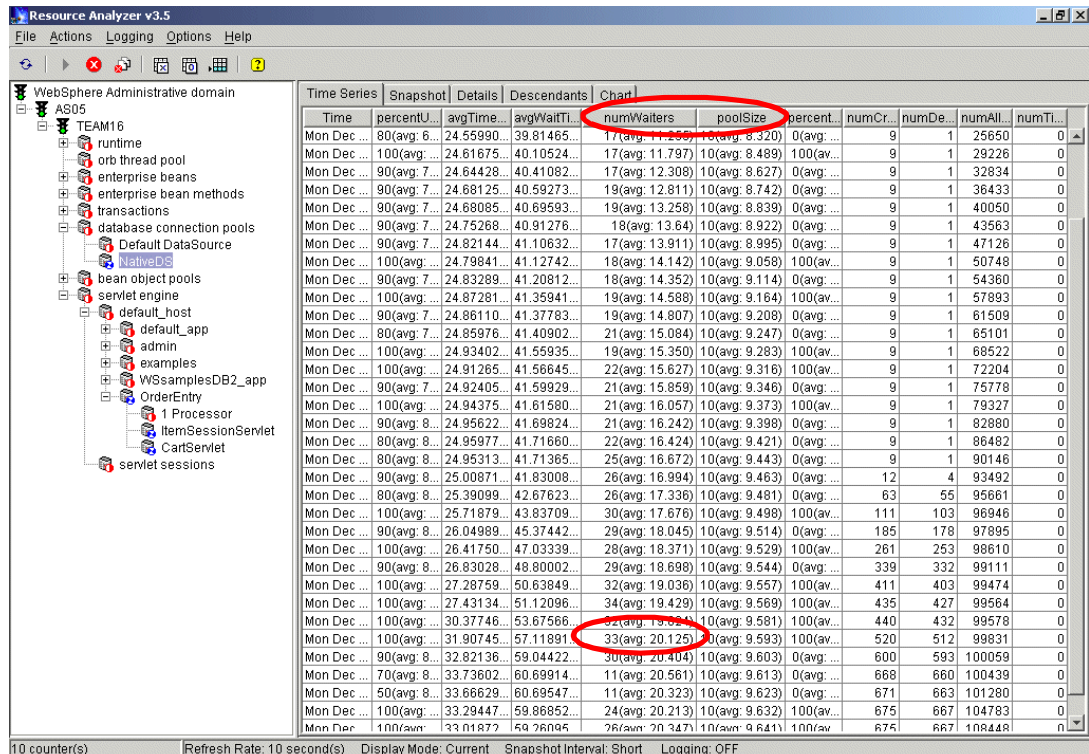


Figure 7-16 Concurrent database connections in the connection pool

The columns *Pool size* and *numWaiters* (circled columns in Figure 7-16) show the number of active connections in this connection pool (in our case 10) and number of connections that are waiting in a given snapshot (maximum 33 in our case) respectively. This tell us that there are many more database connections waiting to be processed than actually running (33 versus 10).

5. We want to determine how many HTTP server threads run concurrently. We start the HTTP server Monitor as described in “Monitoring the number of active threads” on page 90. We see the page shown in Figure 7-17.

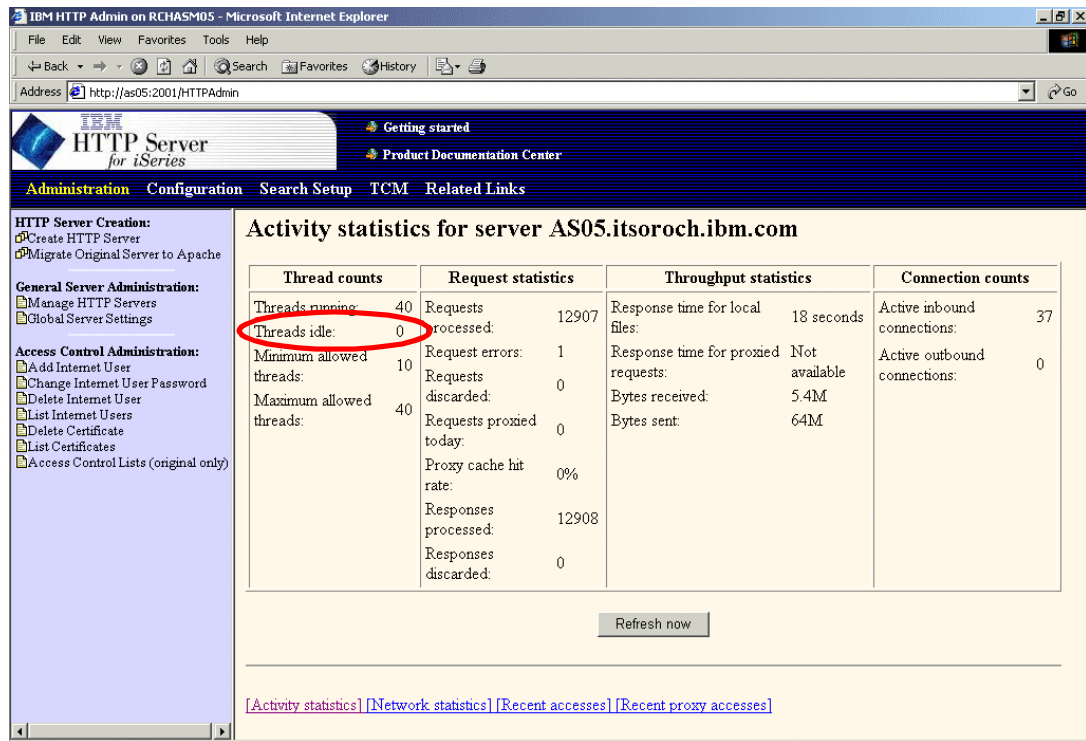


Figure 7-17 Active HTTP server threads

As you can see (as indicated by the arrow), there are no idle threads, which probably means that there are HTTP server threads waiting in the queue.

After consulting the application programmer, they confirmed that clients extensively issue HTTP server requests, which in turn, invoke several servlets and database connections.

After that, we experiment by gradually changing the three queues (HTTP server, servlet engine queue and database connection), always following the funnel principle shown in Figure 5-9 on page 94. However, this doesn't improve our performance. Our CPU is already running at 100%. By increasing the number of active threads, we overwhelm the system (it cannot handle any additional workload), and therefore, actually decrease the performance instead of improving it.

### 7.4.3 Garbage collection initial size

Major improvements have been achieved by gradually increasing the garbage collection initial size to 256 MB from original 32 MB. This way we took some load off of the CPU, since garbage collector ran less frequently, which we found out by using the DMPJVM command (see 3.7.1, “DMPJVM command” on page 40). Increasing the initial size to more than 256 MB did not bring us additional improvements.

## 7.4.4 Other tuning techniques

Disabling auto reload as well as gradually increasing the OSE refresh interval up to 300 brought additional performance. Using HTTP Server (powered by Apache) did not bring improvements in our particular case, even though the powered by Apache server is, on average, about 22% faster than the original HTTP server. The reason for that is low usage of the HTTP server by the OrderEntry application compared to the other components. As you can see in Figure 7-18, when our workload is using 99.7% of CPU, the HTTP server job is using only 0.3%, and therefore, is a minor contributor to the overall CPU usage. We accessed this display by using the WRKACTJOB command and then sorted it by CPU utilization (see 3.5.4, “WRKACTJOB command” on page 30).

Work with Active Jobs						AS05
						12/03/01 23:31:08
CPU %:	99.7	Elapsed time:	00:00:01	Active jobs:	342	
Opt	Subsystem/Job	User	Type	CPU %	Function	Status
	TEAM16	QEJB	BCI	81.5	PGM-QEJBSVR	JVAW
	QSQSRVR	QUSER	PJ	1.4		CNDW
	QSQSRVR	QUSER	PJ	1.4		CNDW
	QSQSRVR	QUSER	PJ	1.4	* -COMMIT	RUN
	QSQSRVR	QUSER	PJ	1.4		CNDW
	QSQSRVR	QUSER	PJ	1.4		CNDW
	QSQSRVR	QUSER	PJ	1.3		CNDW
	QSQSRVR	QUSER	PJ	1.3		CNDW
	QSQSRVR	QUSER	PJ	1.3		CNDW
	QSQSRVR	QUSER	PJ	1.3	* -COMMIT	RUN
	QSQSRVR	QUSER	PJ	1.3		CNDW
	QPADEV0003	ACOVID	INT	.5	CMD-WRKACTJOB	RUN
	TEAM16	QTMHHTTP	BCH	.3	PGM-QZHBHTTP	CNDW
	QDIRSRV	QDIRSRV	BCH	.2	PGM-QGLDSVR	SIGW
	WPL01	QTMHHTTP	BCI	.0	PGM-QZHBHJOB	TIMW
	WPL01	QTMHHTTP	BCI	.0	PGM-QZHBHJOB	TIMW
						More...
==>						
F21=Display instructions/keys						

Figure 7-18 Workload components by CPU utilization

## 7.5 Tuning the application

In this section of the chapter, we analyze the application itself. We use PTDV to identify potential problem areas, and we examine the application code. Our goal is to improve the application throughput. The OrderEntry application uses a combination of servlets and Enterprise JavaBeans.

As shown in Figure 7-8 on page 136, two servlets are used. The ItemSessionServlet servlet displays a list of items available to order, and the CartServlet servlet displays a shopping cart of selected items. The CartServlet servlet also places orders. The servlets use Enterprise JavaBeans to access database tables and to execute order processing. The servlets use a helper class named *ItemsDb* to make accessing EJBs easier and to re-use code. See 7.2, “Application used in this case study” on page 128, for more details about the OrderEntry application.

## 7.5.1 Analyzing the application

We start with the original application. Rather than trying to read through the code, we use PTDV to identify potential problem areas.

Figure 7-19 shows the PTDV cumulative procedure information for one run of the OrderEntry application. Notice that there are inline and cumulative versions of some columns. For example, there are the *Inline CPU Time* and *Cumulative CPU time* columns. Inline refers to the time spent in this method only, while cumulative refers to the time spent in this method and all the methods called by this method.

The screenshot shows the PTDV cumulative procedure information table. Annotations highlight specific areas of interest:

- doGet and doPost methods:** Points to the first two rows of the table.
- EJSJDBCPersistItemBean load and excessive CPU utilization:** Points to the row for `JITC.com-ibm-itso-roch-wasaejb-EJSJDBCPersistItemBean-load`.
- getRangeItemsx method showing high CPU utilization:** Points to the row for `JITC.Support-ItemsDb-getRangeItemsx()`.
- getter methods creating a large number of objects:** Points to the `Inline Object Creat...` column.

Procedure Name	# Invocations	Inline CPU Time	Cumulative CPU	Inline Object Creat	Cumulative ...
JITC.tservlets-ItemSessionServlet-doGet(Ljavax-servlet-http-HttpServletRequest;Ljavax...	1	2	2,966,879	0	333,932
JITC.tservlets-ItemSessionServlet-doPost(Ljavax-servlet-http-HttpServletRequest;Ljavax...	1	1,167	2,968,046	148	333,932
JITC.Support-ItemsDb-getRangeItemsx()Ljava-util-Vector;	1	6,395	2,959,676	1,471	332,069
JITC.com-ibm-itso-roch-wasaejb-EJSJDBCPersistItemBean-load(Ljavax-ejb-EntityBe...	500	953,640	1,041,373	157,999	170,304
JITC.com-ibm-itso-roch-wasaejb-Item_Stub-getItemID()Ljava-lang-String;	100	320	621,266	0	69,561
JITC.com-ibm-itso-roch-wasaejb-ItemHome_Stub-findByPrimaryKey(Lcom-ibm-itso-r...	100	282	611,205	0	70,366
JITC.com-ibm-itso-roch-wasaejb-ItemHome_BaseStub-findByPrimaryKey(Lcom-ibm-i...	100	30,578	610,881	5,801	70,366
JITC.com-ibm-itso-roch-wasaejb-Item_BaseStub-getItemID()Ljava-lang-String;	100	1,238	597,744	0	63,061
JITC.com-ibm-itso-roch-wasaejb-EJSRemoteItem-getItemID()Ljava-lang-String;	100	32,054	596,506	26,700	63,061
JITC.com-ibm-itso-roch-wasaejb-Item_Stub-getItemPrice()F	100	313	589,657	0	64,361
JITC.com-ibm-itso-roch-wasaejb-Item_BaseStub-getItemPrice()F	100	1,350	589,310	0	64,361
JITC.com-ibm-itso-roch-wasaejb-EJSRemoteItem-getItemPrice()F	100	32,262	587,960	26,700	64,361
JITC.com-ibm-itso-roch-wasaejb-EJSRemoteItemHome-findByPrimaryKey(Lcom-ibm-it...	100	265,129	579,699	18,103	64,565
JITC.com-ibm-itso-roch-wasaejb-Item_Stub-getItemInfo()Ljava-lang-String;	100	266	568,874	0	63,061
JITC.com-ibm-itso-roch-wasaejb-Item_BaseStub-getItemInfo()Ljava-lang-String;	100	1,262	568,574	0	63,061
JITC.com-ibm-itso-roch-wasaejb-EJSRemoteItem-getItemInfo()Ljava-lang-String;	100	36,945	567,311	26,700	63,061
JITC.com-ibm-itso-roch-wasaejb-Item_Stub-getItemName()Ljava-lang-String;	100	282	560,257	0	63,061

Figure 7-19 PTDV cumulative procedure information

We see the `doGet` and `doPost` methods for the servlets at the top of the list. This is as expected since the `doGet` and `doPost` methods are always called to handle servlet client requests. You can see that `doGet` uses very few resources. If we review the code, we see that `doGet` simply passes any calls it receives to the `doPost` method. This way, the business logic can all be controlled from one method.

However, the `getRangeItemsx` method, which is part of the `ItemsDb` class, shows a very high CPU utilization. It also shows a large number of object creates. It is a method that we need to investigate further.

The second thing that the PTDV output shows is the large number of EJB-related method objects creates. Notice the `EJSJDBCPersistItemBean` load and the getter methods associated with the `Item` table. The `Item` EJB is a container managed entity bean used to access the `Item` table. It appears that the application creates a large number of objects that are associated with accessing the `Item` table through the `Item` EJB.

### Reviewing the code

Let's examine the code of the `getRangeItemsx` method.

Figure 7-20 shows the `getRangeItemsx` method. This method is passed in a parameter containing an item number. It returns a `Vector` containing a `String` arrays for one hundred rows from the `Items` table, starting with the row passed in as the input parameter.

The `getRangeItemsx` method locates the `Home` interface of the `Item` Enterprise JavaBean. Then it uses the `Home` interface `findByPrimaryKey` method to retrieve rows from the `Item` table.

```
public java.util.Vector getRangeItemsx(int low) {
    Vector itemVector = new Vector();
    try {
        Context ctx = getInitialContext();
        java.lang.Object tempObject = ctx.lookup("Item");
        ItemHome home =
            (ItemHome) javax.rmi.PortableRemoteObject.narrow(
                (org.omg.CORBA.Object) tempObject,
                com.ibm.itso.roch.wasaejb.ItemHome.class);
        for (int count = 0; count < 100 ; count++) {
            String lowString = Integer.toString(low);
            ItemKey itemID = new ItemKey(lowString);
            com.ibm.itso.roch.wasaejb.Item itemValue =
                (com.ibm.itso.roch.wasaejb.Item) home.findByPrimaryKey(itemID);
            String itemString[] = new String[5];
            itemString[0] = itemValue.getItemID();
            itemString[1] = itemValue.getItemName();
            String price = "$" + (itemValue.getItemPrice());
            // make sure the price ends with .00 for even dollar amounts
            if (price.endsWith(".0")) {
                price += "0";
            }
            itemString[2] = price;
            itemString[3] = itemValue.getItemInfo();
            itemVector.addElement(itemString);
            low = Integer.parseInt(lowString);
            low++;
        }; // end while
    } catch (Exception e) {
        System.out.println("::::::::::::: Unexpected Error :::::::::::::::");
        e.printStackTrace();
    }
    return itemVector;
}
```

Figure 7-20 The `getRangeItemsx` method

Two object oriented programming tenants are violated here. First, you should not call an entity bean from a client program. A client program should only interface with a session bean. The second problem is using an entity bean to retrieve a large number of database table rows. In this case, we call the entity bean `findByPrimaryKey` method one hundred times to retrieve one hundred rows. This results in many object creations, database I/Os, and communication flows.

### Fixing the problem

We make two changes to improve the code and the throughput of the application:

- ▶ We change the servlet to interface to the `OrderEntryClerk` session EJB.
- ▶ We use a JDBC call to retrieve the rows from the `Items` table instead of the entity bean.

We change the ItemSessionServlet servlet to call the findRangeOfItems method in the ItemsDb class. As shown in Figure 7-21, it calls the findRangeOfItems method of the OrderEntryClerk EJB.

```
public Vector findRangeOfItems(String min, String max) {
    Vector ejbItems = null;
    try {
        ejbItems = clerk.findRangeOfItems(min, max);
    } catch (Exception e) {
        System.out.println(":findRange of Items Unexpected Error " + e.getMessage());
        //e.printStackTrace();
    }
    finally {
        return ejbItems;
    }
}
```

Figure 7-21 The ItemsDb findRangeOfItems method

Figure 7-22 shows the findRangeOfItems method of the OrderEntryClerk EJB. It passes two parameters that contain the range of Item rows to return. It creates a JDBC statement object and uses it to retrieve rows from the Item table and the Stock table. It returns a Vector that contains a String array for each row returned.

```
public Vector findRangeOfItems(String lowID, String highID) throws RemoteException {
    Vector itemVector = new Vector();
    Item item = null;
    try {
        Connection con = ds.getConnection();
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("select iid, iname, iprice, idata, stqty from item, " +
            "stock where iid = stiid and iid >= '" + lowID +
            "' and iid <= '" + highID + "'");
        while ( rs != null && rs.next() ) {
            String itemString[] = new String[5];
            itemString[0] = rs.getString("iid");
            itemString[1] = rs.getString("iname");
            itemString[2] = "$" + Float.toString(rs.getFloat("iprice"));
            itemString[3] = Integer.toString(rs.getInt("stqty"));
            itemString[4] = rs.getString("idata");
            itemVector.addElement(itemString);
        }
        stmt.close();
        releaseConnection(con);
    } catch (Exception e) {
        throw new RemoteException(e.getMessage());
    }
    return itemVector;
}
```

Figure 7-22 The OrderEntryClerk findRangeOfItems method

## The results

We implement the change and rerun the application. In our test, we use two hundred clients to place five orders each, for a total of one thousand orders.

As shown in Table 7-10, we see a dramatic improvement in the application. In the original application, it took over thirty one minutes to place one thousand orders. In the new application, it takes less than eight minutes to do the same work.

Table 7-10 Comparing the applications

Application	Orders placed	Elapsed time	CPU utilization
Original	1000	31 minutes 9 seconds	100%
Improved	1000	7 minutes 49 seconds	35%

Let's look at the output from PTDV for the new application as shown in Figure 7-23. The first thing to notice is that the CPU time is much less. The doPost method now has a cumulative CPU Time of 59,888 milliseconds (left circle in the figure). Before it was almost 3,000,000 milliseconds.

We also create fewer objects. The doPost method now shows Cumulative Object Creates equal to 7,785 (right circle in the figure). Previously this was 333,392. The other methods show corresponding decreases in CPU time and object creates. This is because we no longer create a large number of instances of the Item entity bean. Now most of the processing is done in the OrderEntryClerk and OrderPlacement EJBs.

Procedures called:					
Procedure Name	# Invocations	Inline CPU Time	Cumulative CPU	Inline Object	Cumulative
<unknown>.<unknown>	12	65,998	179,704	1,747	26,618
JITC.servlets-CartServlet-doPost(Ljavax-servlet-http-HttpServletRequest;Ljavax-servlet-http-...	3	3,051	59,888	643	7,785
JITC.servlets-CartServlet-placeOrder(Ljava-io-PrintWriter;LshopData-ShoppingCart;Ljava-la...	1	1,063	55,057	204	6,991
JITC.servlets-ItemSessionServlet-doGet(Ljavax-servlet-http-HttpServletRequest;Ljavax-servl...	1	5	52,814	0	16,096
JITC.servlets-ItemSessionServlet-doPost(Ljavax-servlet-http-HttpServletRequest;Ljavax-serv...	1	1,181	52,809	236	16,096
JITC.com-ibm-itso-roch-wasaejb-_OrderEntryClerk_Stub-findRangeOfItems(Ljava-lang-Strin...	1	3	37,511	0	13,302
JITC.com-ibm-itso-roch-wasaejb-_OrderEntryClerk_BaseStub-findRangeOfItems(Ljava-lang...	1	14,903	37,249	8,466	13,239
JITC.com-ibm-itso-roch-wasaejb-_OrderPlacement_Stub-placeOrder(Ljava-lang-String;LJa...	1	4	34,655	0	5,050
JITC.com-ibm-itso-roch-wasaejb-_OrderPlacement_BaseStub-placeOrder(Ljava-lang-Strin...	1	694	34,650	271	5,050
JITC.com-ibm-itso-roch-wasaejb-EJSRemoteOrderPlacement-placeOrder(Ljava-lang-Strin...	1	6,135	33,955	517	4,779
JITC.com-ibm-itso-roch-wasaejb-OrderPlacementBean-placeOrder(Ljava-lang-String;LJa...	1	1,866	24,800	338	3,781
JITC.com-ibm-itso-roch-wasaejb-EJSRemoteOrderEntryClerk-findRangeOfItems(Ljava-lang-...	1	3,101	22,346	185	4,773
JITC.com-ibm-itso-roch-wasaejb-OrderEntryClerkBean-findRangeOfItems(Ljava-lang-String;...	1	19,227	19,245	4,338	4,588
JITC.com-ibm-itso-roch-wasaejb-_CustomerHome_Stub-findByPrimaryKey(Lcom-ibm-itso-r...	1	3	10,199	0	1,474
JITC.com-ibm-itso-roch-wasaejb-_CustomerHome_BaseStub-findByPrimaryKey(Lcom-ibm-...	1	277	10,196	58	1,474
JITC.com-ibm-itso-roch-wasaejb-EJSRemoteCustomerHome-findByPrimaryKey(Lcom-ibm-i...	1	270	9,913	4	1,416
JITC.com-ibm-itso-roch-wasaejb-EJSCustomerHomeBean-findByPrimaryKey(Lcom-ibm-its...	1	1,185	9,642	161	1,412
JITC.com-ibm-itso-roch-wasaejb-_DistrictHome_Stub-findByPrimaryKey(Lcom-ibm-itso-roch...	1	3	8,310	0	726
JITC.com-ibm-itso-roch-wasaejb-_DistrictHome_BaseStub-findByPrimaryKey(Lcom-ibm-its...	1	508	8,307	91	726
JITC.com-ibm-itso-roch-wasaejb-EJSRemoteDistrictHome-findByPrimaryKey(Lcom-ibm-itso...	1	2,728	7,796	181	635
JITC.com-ibm-itso-roch-wasaejb-CustomerBean-refresh(Lcom-ibm-itso-roch-wasaejb-Cust...	2	7,093	7,111	994	994
JITC.com-ibm-itso-roch-wasaejb-_District_Stub-getNextOrderId(Z)	1	3	6,866	0	702
JITC.com-ibm-itso-roch-wasaejb-_District_BaseStub-getNextOrderId(Z)	1	18	6,861	0	702
JITC.com-ibm-itso-roch-wasaejb-EJSJDBCPersisterDistrictBean-load(Ljavax-ejb-EntityBean...	2	5,531	6,853	643	755
JITC.com-ibm-itso-roch-wasaejb-EJSRemoteDistrict-getNextOrderId(Z)	1	3,678	6,843	267	702
JITC.servlets-ItemSessionServlet-outputItemInformation(Ljava-io-PrintWriter;Ljava-util-Vecto...	1	5,561	5,718	1,720	1,720

Figure 7-23 PTDV output for improved application



## 7.5.2 Further improving the application

So far, we made one change that made a dramatic improvement in the application. It was also a change that was fairly easy to implement. This is known as “picking the low hanging fruit”. Next, we work to improve the application further. Now we have to work harder to gain small improvements. However, if we find enough of these small improvements, we can again significantly improve the application performance. This is known as “harvesting the high hanging fruit”.

### Caching EJB Home interfaces

In order to use an Enterprise Java Bean, you must locate it on the server. This is done through its Home interface. EJB homes are obtained from WebSphere Application Server through a JNDI naming lookup. This is an expensive operation that can be minimized by caching and reusing EJB Home objects. For simple applications, it might be enough to acquire the EJB home in the servlet `init()` method. More complicated applications might require cached EJB homes in many servlets and EJBs. In the `OrderEntry` application, the servlets use two session beans. They are the `OrderEntryClerk` and `OrderPlacement` enterprise beans.

The `ItemSessionServlet` class uses the `OrderEntryClerk` EJB to obtain a list of items to display to the end user. Before we can use an EJB, we must obtain its Home interface. If we successfully obtain the Home interface, we can use it to create an instance of the EJB. We can then use the instance of the bean to perform business logic.

As shown in Figure 7-24, the Home interface for the `OrderEntryClerk` EJB is obtained in the `doPost` method in the `ItemSessionServlet` class. The Home interface is used to create an instance of the `OrderEntryClerk` enterprise bean, which we name *clerk*. We then call the `findRangeOfItems` method of the `OrderEntryClerk` bean instance to obtain the list of items to display.

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws IOException {
    OrderEntryClerkHome oohome = null;
    OrderEntryClerk clerk = null;
    try {
        Context ctx = MyItemsDB.getInitialContext();
        java.lang.Object tempObject = ctx.lookup("OrderEntryClerk");
        oohome =
            (OrderEntryClerkHome) javax.rmi.PortableRemoteObject.narrow(
                (org.omg.CORBA.Object) tempObject,
                com.ibm.itso.roch.wasaejb.OrderEntryClerkHome.class);
        clerk = (OrderEntryClerk) oohome.create();
    }

    } catch (Throwable t) {

        t.printStackTrace();
        flexLog("error connecting!");
        throw new IOException("Could not connect, cannot continue");

    }

    Vector items = clerk.findRangeOfItems(get1, get2);
```

Figure 7-24 *ItemSessionServlet* `doPost` method

Although this method of obtaining the Home interface works, it can cause a performance problem. This is because each time a user runs the `ItemSessionServlet`, the `doPost` method is executed. If we have hundreds or even thousands of users, we look up the Home interface for each one. To solve this problem, we move the look up of the Home interface to the servlet `init` method.

Use the `HttpServlet init()` method to perform expensive operations that need only be done *once*.

Because the servlet `init()` method is invoked when servlet instance is loaded, it is the perfect location to carry out expensive operations that only need to be performed during initialization. By definition, the `init()` method is thread-safe. The result of operations in the `HttpServlet.init()` method can be cached safely in servlet instance variables, which become read-only in the servlet service method.

The servlets use a helper class, named `ItemsDb`. As shown in Figure 7-25, we add class variables to hold values for the `OrderEntryClerk` and `OrderPlacement` Home interfaces.

```
public class ItemsDb extends java.lang.Object {
    private static String systemName = new String("");
    private static String userid = new String("");
    private static String password = new String("");
    private static String port = new String("");
    public static OrderEntryClerkHome oohome = null;
    public static OrderPlacementHome ophome = null;
}
```

Figure 7-25 *ItemsDb* class variables

We add a new method, named `init`, to the `ItemsDb` class. As shown in Figure 7-26, in the `init` method, we look up the Home interfaces for both the `OrderEntryClerk` and `OrderPlacement` enterprise beans and store their values in the class variables.

```
public void init() {
    try {
        Context ctx = getInitialContext();
        java.lang.Object tempObject = ctx.lookup("OrderEntryClerk");
        oohome =
            (OrderEntryClerkHome) javax.rmi.PortableRemoteObject.narrow(
                (org.omg.CORBA.Object) tempObject,
                com.ibm.itso.roch.wasaejb.OrderEntryClerkHome.class);
        tempObject= ctx.lookup("OrderPlacement");

        ophome = (OrderPlacementHome)javax.rmi.PortableRemoteObject.narrow(
            (org.omg.CORBA.Object) tempObject, com.ibm.itso.roch.wasaejb.OrderPlacementHome.class);

    } catch (Exception e) {
        System.out.println("Could not connect " + e.getMessage());
        e.printStackTrace();
    } catch (Throwable t) {
        t.printStackTrace();
    }
}
```

Figure 7-26 *ItemsDb* `init` method

Next, we update the `init` method of the `ItemSessionServlet` class. As shown in Figure 7-27, we create an instance of the `ItemsDb` class, set some variables, and call its `init` method. The class variables in the `ItemsDb` instance now contain the Home interfaces for the `OrderEntryClerk` and `OrderPlacement` enterprise beans.

```
MyItemsDB = new Support.ItemsDb();

MyItemsDB.setPassword(password);
MyItemsDB.setUserId(userid);
MyItemsDB.setSystemName(system);
MyItemsDB.setPort(port);
MyItemsDB.init();
```

Figure 7-27 *ItemSessionServlet init method*

As shown in Figure 7-28, we change the `ItemSessionServlet doPost` method to use the cached Home interface to create an instance of the `OrderEntryClerk` EJB. This change can save considerable processing time. If the `ItemSessionServlet` servlet is called one thousand times, we now look up the home interface once, instead of one thousand times.

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws IOException {

    OrderEntryClerk clerk = null;
    try {
        clerk = (OrderEntryClerk) MyItemsDB.oohome.create();
    }
}
```

Figure 7-28 *ItemSessionServlet doPost method*

We make the same changes to the `CartServlet` class. In the `CartServlet` class, we use both the `OrderEntryClerk` and `OrderPlacement` enterprise beans, so the effects of this change can be even more dramatic.

## Removing instances of stateful session beans

Instances of stateful session beans have affinity to specific clients. They remain in the container until they are explicitly removed by the client or are removed by the container when they time-out. Meanwhile, the container might need to passivate inactive stateful session beans to disk. This requires overhead for the container and constitutes a performance hit to the application. If the passivated session bean is subsequently required by the application, the container activates it by restoring it from disk. By explicitly removing stateful session beans when finished with them, applications decrease the need for passivation and minimize container overhead. The `OrderEntryClerk` enterprise bean is a stateful session bean. To remove a stateful session bean, we use:

```
clerk.remove();
```

## Database access using JDBC

When using JDBC to access an iSeries database table, there are a number of performance enhancing techniques that you can use. Recall that one of the first changes we made to the application was to use JDBC to access the database to retrieve a large number of records. We changed the `ItemSessionServlet` class to execute the `findRangeOfItems` method of the `OrderEntryClerk` EJB. The code for the `findRangeOfItems` method is shown in Figure 7-22 on page 153.

### ***Prepared statements***

Even with the statement cache, it is still necessary for performance reasons to prepare (prepareStatement) SQL statements prior to executing (executeQuery or executeUpdate methods) them repeatedly. If the repeatable statement is not prepared beforehand, the executeQuery causes an implicit prepareStatement to occur every time the statement is run. Therefore, the overhead will be higher without an explicit prepare.

You prepare the statement once using the prepareStatement() method on the Connection object and then execute it whenever required using execute(), executeQuery(), or executeUpdate() on the PreparedStatement object.

### ***Retrieving only the necessary columns***

It is very common to write SQL statements making use of the syntax:

```
select * from employees where....
```

This has the side effect of returning every single column in the file in the result set. A much faster and, in effect, just as easy technique to use is to select the specific columns that are needed using the following syntax:

```
select firstname, lastname, department from employees where...
```

This has the effect of only bringing back the columns that are needed and can also reduce the number of EBCDIC to Unicode conversions taking place if the data is stored in EBCDIC.

### ***Retrieving by index rather than name***

Getting column data by index instead of name is faster, since the JDBC driver does not have to perform name resolution. Note that on some platforms, using a column index requires the get methods to be performed in column sequence. This is not a restriction on the iSeries server.

Figure 7-29 shows the updated findRangeOfItems method. We change the method to use a prepared statement. Previously, the method executed a join between the Item table and Stock table, so the quantity in stock value could be retrieved. This value is not required for the creation of the output table, so we eliminate the join operation. We also do not retrieve the description from the Item table, since it is not used. Finally, we retrieve the required values by index rather than name.

We make similar changes to the application wherever JDBC is used.

```

public Vector findRangeOfItems(String lowID, String highID) throws RemoteException {
    Vector itemVector = new Vector();
    Item item = null;
    try {
        Connection con = ds.getConnection();
        PreparedStatement psAllRecord = null;

        psAllRecord = con.prepareStatement("select iid, iname, iprice from item where iid >= ? and iid
            <= ? ");
        psAllRecord.setString(1, lowID);
        psAllRecord.setString(2, highID);

        ResultSet rs = psAllRecord.executeQuery();
        while ( rs != null && rs.next() ) {
            String itemString[] = new String[5];
            itemString[0] = rs.getString(1);
            itemString[1] = rs.getString(2);
            itemString[2] = "$" + Float.toString(rs.getFloat(3));
            itemVector.addElement(itemString);
        }
        psAllRecord.close();
        releaseConnection(con);
    } catch (Exception e) {
        throw new RemoteException(e.getMessage());
    }
    return itemVector;
}

```

Figure 7-29 Updated findRangeOfItems

## EJB isolation levels and transaction processing

The isolation level of a transaction describes the lengths to which the resource manager will go to provide the *isolation property* of transactions. At the highest levels, the resource manager uses locking and other strategies to ensure that one transaction does not affect the outcome of another transaction until it has been completed. This can drastically affect the performance characteristics of your application.

Consider two example applications. One allows Web users to browse an online catalog. The other continually updates prices and inventory of that catalog.

If the transactions used by each application were completely isolated from each other, performance could conceivably suffer greatly. The reality is that for this particular set of transactions, the isolation requirements for transactions used by the Web users as they look at the catalog are not critical. It is not until the user submits an order that the inventory and price of the item must be guaranteed.

By associating too high of an isolation level with the Web user's transactions, throughput of the Web site could be constrained. On the other hand, if you use too low of an isolation level, your application is susceptible to what appears to be inconsistent or out of date data.

The use of an isolation level allows the application developer to directly take control of the performance versus integrity trade-offs that match the application. Furthermore, in a WebSphere EJB environment, the system administrator can or application deployer can customize the isolation levels used by the application to more accurately match the requirements of the business environment. This customization can occur without any change to the source code or without the application developer being required to take any explicit action. We discuss this in more detail later in this chapter.

### **Possible isolation errors**

At various isolation levels, there exists the possibility of the following transaction isolation phenomena. The fact that these phenomena can occur is intended to allow the database user or developer the ability to choose the desired balance between data integrity or desire for a view of the most up-to-date data versus performances and concurrency.

- ▶ **Dirty read:** A dirty read happens when an application running under one transaction reads data from another transaction that has not yet been committed. The second transaction can roll back its changes. The dirty data read by the first transaction should have never existed because the second transaction rolled back its changes.
- ▶ **Non-repeatable read:** A non-repeatable read happens when an application running under one transaction reads the same data twice, but gets different values for the data each time. A different transaction updates or deletes data and then commits the changes. When the first transaction reads the data again, it encounters a non-repeatable read because different values were read or the data was deleted. Note that, in this case, the first transaction has only read committed data.
- ▶ **Phantom read:** A phantom read happens when an application running under one transaction reads some data while another transaction generates new data (by an insert or update). The next time the first transaction attempts to read the same data, it reads new or phantom data. Note that, in this case, the first transaction has only read committed data, but depending on the search criteria, new data was added that matches the criteria.

### **Transaction isolation attribute**

The transaction isolation attribute limits concurrent reads in a database. The valid values for this attribute in decreasing order of isolation are:

- ▶ TRANSACTION\_SERIALIZABLE
- ▶ TRANSACTION\_REPEATABLE\_READ
- ▶ TRANSACTION\_READ\_COMMITTED
- ▶ TRANSACTION\_READ\_UNCOMMITTED

These isolation levels correspond to the isolation levels defined in the Java Database Connectivity (JDBC) `java.sql.Connection` interface. Table 7-11 shows how the container uses the transaction isolation level attribute.

*Table 7-11 Transaction isolation levels*

Transaction isolation level (SQL and JDBC)	Dirty reads	Non-repeatable reads	Phantom reads
Read uncommitted (lowest no isolation)	Yes	Yes	Yes
Read committed	No	Yes	Yes
Repeatable read	No	No	Yes
Serializable (highest complete isolation)	No	No	No

A transaction can always see its own changes before committing, regardless of the isolation level. The transaction isolation level determines how isolated one transaction is from another. This attribute can be set for individual methods in a bean. However, within a transactional context, the isolation level associated with the first method invocation becomes the required isolation level for all other methods invoked within that transaction. If a method is invoked with a different isolation level than that of the first method, an exception is thrown.

***Trade-off: Performance versus consistency***

If we always use the safe value of TRANSACTION\_SERIALIZABLE, we're 100% isolated, but performance may not be acceptable.

If performance is more important than consistency in your application, you may want to turn off locking for values you're only going to read.

A Phone directory servlet, for example, may not need to lock out changes while it's being read. In this case, you could use TRANSACTION\_READ\_UNCOMMITTED on the methods that perform the search in the session bean and the finder entity bean methods.

The safer or more isolated that you make your transactions, the higher a performance penalty you will pay. If you use the highest level, TRANSACTION\_SERIALIZABLE, you will not be able to even read records when they are part of another transaction. Depending on your application, you may decide that reading dirty or uncommitted rows is acceptable. In this case, your application will not wait on locks set by other transactions. We do not recommend that you use TRANSACTION\_SERIALIZABLE for every method in a bean, because it will lock referenced tables and release them only when the transaction finishes.

For read only entity beans, set the Read-Only attribute to Yes. The container can use this information to optimize database operations. For example, if all methods executed on an entity bean during a transaction are marked as read-only, the container can skip the ejbStore() operation at the end of the transaction. This helps avoid possible read to write lock promotions and improves performance by omitting the database write that can result in exceptions caused by conflicts.

By default, most developers deploy EJBs with the transaction isolation level set to TRANSACTION\_SERIALIZABLE. This used to be the default in IBM VisualAge for Java - Enterprise Edition and still is in other EJB deployment tools. It is also the most restrictive and protected transaction isolation level incurring the most overhead.

Most workloads do not require the isolation level and protection afforded by TRANSACTION\_SERIALIZABLE. This isolation level has a major performance impact and shouldn't be used unless absolutely necessary. A given application might never update the underlying data or be run with other concurrent updaters.

In that case, the application would not have to be concerned with dirty, non-repeatable, or phantom reads. TRANSACTION\_READ\_UNCOMMITTED would probably be sufficient.

Because the EJB's transaction isolation level is set in its deployment descriptor, the same EJB could be reused in different applications with different transaction isolation levels. Review your isolation level requirements and adjust them appropriately to increase performance.

In our application, the OrderEntryClerk EJB does not update any tables. As shown in Figure 7-30, we use the WebSphere console to change the methods of the OrderEntryClerk bean, that we use, to use an isolation level of TRANSACTION\_READ\_UNCOMMITTED.

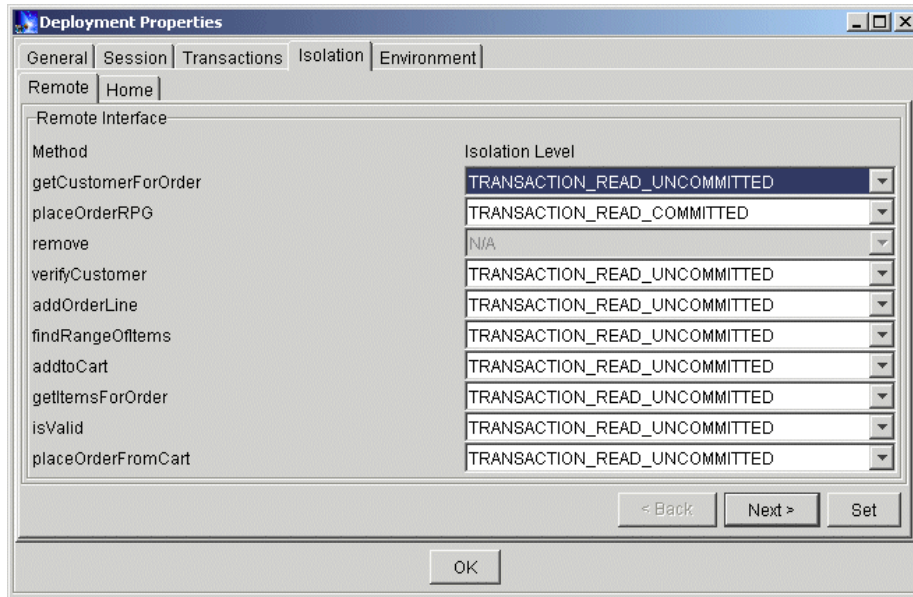


Figure 7-30 Setting the isolation level

As shown in Figure 7-31, we also set the Read-Only attribute on.

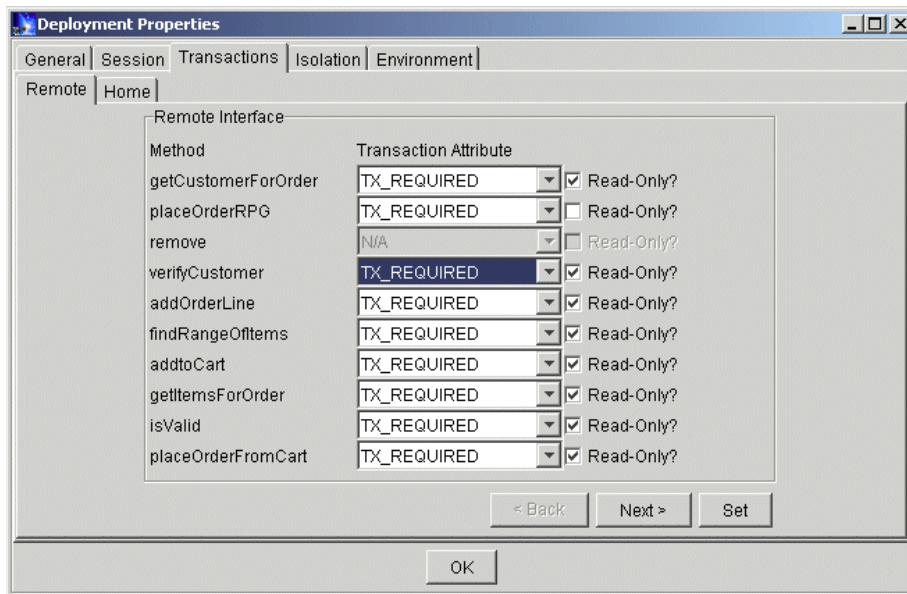


Figure 7-31 Setting the Read-Only attribute

The OrderPlacementBean updates tables. We set the OrderPlacement bean to use an isolation level of TRANSACTION\_REPEATABLE\_READ. We set the Read-Only attribute on where appropriate.

## The results

Here is a summary of the changes made to the application:

- ▶ Cache and re-use EJB Home interfaces
- ▶ Remove stateful session beans when no longer needed
- ▶ Use JDBC prepared statements



- ▶ Select only required columns for SQL statements
- ▶ Retrieve SQL columns by index rather than name
- ▶ Use less restrictive isolation levels for EJBs
- ▶ Set the read only transaction attribute for EJB methods where appropriate

We implement the changes and re-run the application. In our test, we use two hundred clients to place five orders each, for a total of one thousand orders. The results are shown in Table 7-12.

Table 7-12 Comparing the applications

Application	Orders placed	Elapsed time	CPU utilization
Original	1000	31 minutes 9 seconds	100%
Improved	1000	7 minutes 49 seconds	42%
Improved Again	1000	6 minutes 13 seconds	38%

As shown in Table 7-12, we see an improvement in the application. The improvement is not as dramatic as our first improvement, but it is still significant.

Let's look at the output from PTDV for the new application as shown in Figure 7-32.

Procedures called:						
Procedure Name	# Invocations	Inline CPU Tim...	Cumulative CP...	Inline Object Creat...	Cumul...	
JITC.tservlets-CartServlet-doPost(Ljavax-servlet-http-HttpServletRequest;Ljavax-servlet-h...	3	3,497	60,546	643	2,631	
JITC.tservlets-CartServlet-placeOrder(Ljava-io-PrintWriter;LshopData-ShoppingCart;Ljav...	1	657	55,537	39	5,817	
JITC.tservlets-ItemSessionServlet-doGet(Ljavax-servlet-http-HttpServletRequest;Ljavax-s...	1	2	42,910	0	11,543	
JITC.tservlets-ItemSessionServlet-doPost(Ljavax-servlet-http-HttpServletRequest;Ljavac...	1	917	42,908	132	11,543	
JITC.com-ibm-itso-roch-wasaejb-_OrderPlacement_Stub-placeOrderx(Ljava-lang-String;...	1	4	35,693	0	5,134	
JITC.com-ibm-itso-roch-wasaejb-_OrderPlacement_BaseStub-placeOrderx(Ljava-lang-...	1	769	35,689	271	5,134	
JITC.com-ibm-itso-roch-wasaejb-EJSSRemoteOrderPlacement-placeOrderx(Ljava-lang-S...	1	6,581	34,919	532	4,863	
JITC.com-ibm-itso-roch-wasaejb-_OrderEntryClerk_Stub-findRangeOfItems(Ljava-lang-...	1	3	27,383	0	9,040	
JITC.com-ibm-itso-roch-wasaejb-_OrderEntryClerk_BaseStub-findRangeOfItems(Ljava-l...	1	11,883	27,136	6,950	8,975	
JITC.com-ibm-itso-roch-wasaejb-OrderPlacementBean-placeOrderx(Ljava-lang-String;L...	1	2,222	24,692	338	3,819	
JITC.com-ibm-itso-roch-wasaejb-EJSSRemoteOrderEntryClerk-findRangeOfItems(Ljava-l...	1	3,189	15,253	185	2,025	
JITC.com-ibm-itso-roch-wasaejb-OrderEntryClerkBean-findRangeOfItems(Ljava-lang-St...	1	12,048	12,064	1,840	1,840	
JITC.com-ibm-itso-roch-wasaejb-_DistrictHome_Stub-findByPrimaryKey(Lcom-ibm-itso-...	1	3	8,299	0	717	
JITC.com-ibm-itso-roch-wasaejb-_DistrictHome_BaseStub-findByPrimaryKey(Lcom-ibm-...	1	482	8,295	91	717	
JITC.com-ibm-itso-roch-wasaejb-_CustomerHome_Stub-findByPrimaryKey(Lcom-ibm-it...	1	3	8,258	0	1,383	
JITC.com-ibm-itso-roch-wasaejb-_CustomerHome_BaseStub-findByPrimaryKey(Lcom-i...	1	282	8,254	58	1,383	
JITC.com-ibm-itso-roch-wasaejb-EJSSRemoteCustomerHome-findByPrimaryKey(Lcom-i...	1	272	7,966	4	1,325	
JITC.com-ibm-itso-roch-wasaejb-EJSSRemoteDistrictHome-findByPrimaryKey(Lcom-ibm-...	1	2,788	7,811	181	626	
JITC.com-ibm-itso-roch-wasaejb-EJSSCustomerHomeBean-findByPrimaryKey(Lcom-ibm-...	1	1,168	7,694	161	1,321	

Figure 7-32 PTDV output

Notice the doPost method for the ItemSession servlet. We now see a Cumulative CPU time of 42,908 milliseconds (top circle in the figure) and Cumulative Object Creates of 11,543. If you refer to Figure 7-23 on page 154, you can see that these values were 52,814 and 16,046 respectively.

Next, look at the findRangeOfItems method. The value for Cumulative CPU time is 27,383 milliseconds (bottom circle in the figure) and Cumulative Objects Creates is 9,040. Previously these values were 37,249 and 13,239.

We see such improvement throughout the application. A number of small changes adds up to a significant performance improvement.

## 7.6 Changing to WebSphere 4.0

WebSphere 4.0 became available for the iSeries server in October 2001. One of its major benefits is improved performance. WebSphere 4.0 provides a Java 1.3 implementation. The IBM Java Runtime Environment (JRE) 1.3 provides up to a 22 percent improvement over the JRE 1.2.2. Improvements in the JVM, JIT compiler, garbage collection, and threads management account for the performance improvements.

We deploy the latest version of the application to run under WebSphere Advanced Edition 4.0 and re-run the application. In our test, we use 200 clients to place five orders each, for a total of 1000 orders. The results are shown in Table 7-13.

*Table 7-13 Comparing the applications*

Application	Orders placed	Elapsed time	CPU utilization
Original	1000	31 minutes 9 seconds	100%
Improved	1000	7 minutes 49 seconds	42%
Improved Again	1000	6 minutes 13 seconds	38%
WebSphere 4.0	1000	4 minutes 36 seconds	36%

As shown in Table 7-13, we see an improvement in the application. This improvement is not attributable to any application changes that we made, but to WebSphere Application Server 4.0.

## 7.7 Summary

In this case study, we attempted to show you the methodology and an example of using the available tools to improve the performance of a sample application. We tested many tuning options, some of them brought significant improvements, while others proved to have no influence on the performance in this particular case. You may experience different effects of these tuning techniques in your environment. Figure 7-33 shows the summary of our tuning activities and how much we decreased our run time by tuning the system, then tuning WebSphere Application Server and HTTP server, next tuning the application, and finally running everything on WebSphere Application Server Version 4.0.

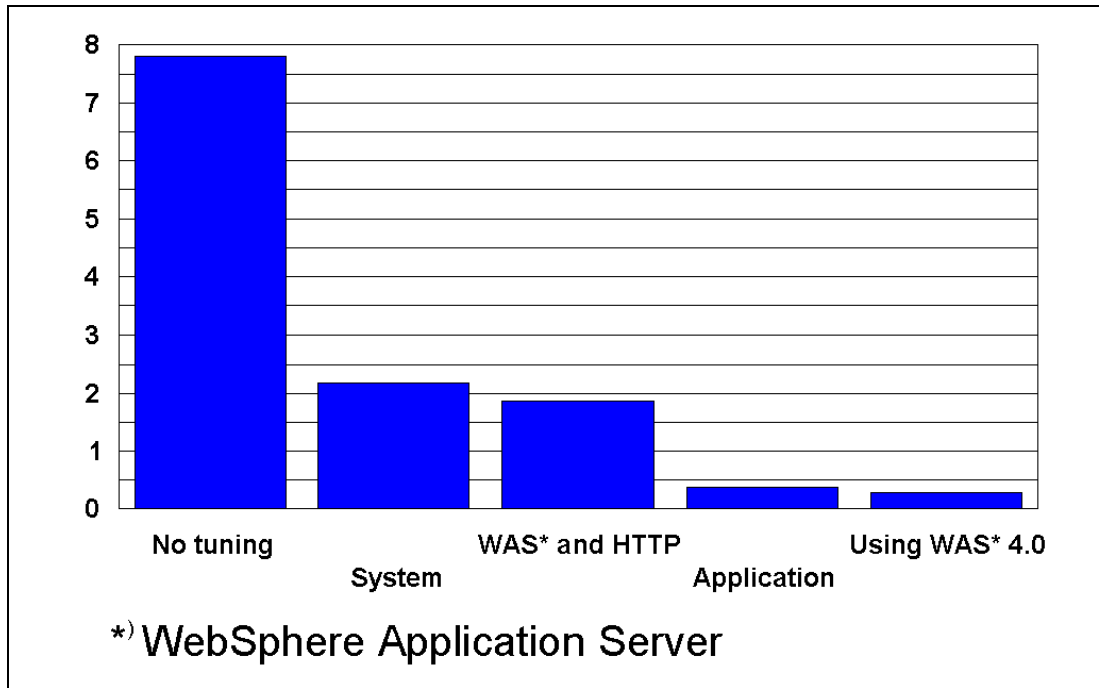


Figure 7-33 Effect of tuning techniques





## Scaling the WebSphere environment

This chapter considers the aspects of scaling a WebSphere application. It starts by summarizing the general factors that should be considered when looking at scaling an application. Then it describes in greater detail the specific iSeries characteristics that assist in achieving a highly scalable application environment.

## 8.1 Scalability objectives

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

- **Scalability:** The proposed configurations should allow the overall system to service a higher client load than that provided by the simple basic configuration. Ideally, it should be possible to service any given load, simply by adding the appropriate number of servers or machines.
- **Load balancing:** The proposed configurations 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 would not do for one machine to be overloaded, while another machine is mostly idle. If all machines are of roughly the same power, each should process a roughly equal share of the load. If various machines are of different power, each should process a portion of the load in proportion to its relative processing power.

Furthermore, if the total load changes over time, the system should naturally adapt itself to maintain this load-balancing property, regardless of the actual total magnitude of this load. In other words, all machines may be used at 50% of their capacity, or all machines may be used at 100% of their capacity.

- **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 is redistributed to the remaining N-1 servers, each of which will process a proportionately slightly higher percentage of the total load. Of course, such an arrangement pre-supposes that the system is designed with some degree of over-capacity, so that N-1 servers are indeed sufficient to process the total expected client load.

Ideally, the failover aspect should be totally transparent to clients of the system. When a server fails, any client that is currently interacting with that server should be automatically redirected to one of the remaining servers, without any interruption of service and without requiring any special action on the part of that client. In practice, however, most failover solutions may not be completely transparent.

For example, a client that is currently in the middle of an operation when a server that fails may receive an error from that operation and may be required to retry (at which point the client would be connected to a different, still-available server). Or the client may observe a “hiccup” or delay in processing, before the processing of their requests resumes automatically with a different server.

The important point for failover, is that each client, and the set of clients as a whole, can eventually continue to take advantage of the system and receive useful service, even if some of the servers fail and become unavailable. Conversely, when a previously-failed server is repaired and again becomes available, the system may transparently start using that server again to process a portion of the total client load.

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

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

In addition, we should also consider a number of secondary characteristics when evaluating a configuration:

- ▶ **Dynamic changes to configuration:** In certain configurations, it may be possible to modify the configuration on-the-fly without interrupting the operation of the system and its service to clients. For example, it may be possible to add or remove servers to adjust to variations in the total client load. Or it may be possible to temporarily stop one server to change some operational or tuning parameters, and then restart it and continue to service client requests. Such characteristics, when possible, are highly desirable, since they enhance the overall manageability and flexibility of the system.
- ▶ **Mixed configuration:** In some configurations, it may be possible to mix multiple versions of a server or application, so as to provide for staged deployment and a smooth upgrade of the overall system from one software or hardware version to another. Coupled with the ability to make dynamic changes to the configuration, this property may be used to effect upgrades without any interruption of service.
- ▶ **Fault isolation:** In the simplest application of failover, we are only concerned with clean failures of an individual server, in which a server simply stops to function completely, but this failure has no effect on the health of other servers. But there are sometimes situations where one malfunctioning server may, in turn, create problems for the operation of other, otherwise healthy, servers. For example, one malfunctioning server may hoard system resources, or database resources. Or it may hold critical shared objects for extended periods of time, and prevent other servers from getting their fair share of access to these resources or objects. In this context, some configurations may provide a degree of fault isolation, in that they reduce the potential for the failure of one server to affect other servers.
- ▶ **Security:** The potential to distribute the processing responsibilities between multiple servers and, in particular, multiple machines, also introduces a number of opportunities for meeting special security constraints in the system. Different servers or types of servers may be assigned to manipulate different classes of data or perform different types of operations. And the interactions between the various servers may be controlled – for example through the use of firewalls – to prevent undesired accesses to data. This book does not specifically focus on security and firewall-related issues, but tries to point out the important observations as they pertain to various configurations.

## 8.2 Scaling a solution

This section looks at various aspects of scalability. It starts with workload patterns and continues by looking at the steps that we see as relevant to scaling your application.

### 8.2.1 Workload patterns

Most high-volume Web sites experience volumes that can be characterized as “bursty”, meaning very high and very low volumes in any 24-hour period, with as much as 5 or 10 times the variation.

Such “burstiness” underscores the challenge of planning for volumes and suggests that planning for average volumes is ineffective. As e-businesses serve more and more international users, this burstiness factor may level out some, but most likely at moderate to high volumes.

Other factors that define the workload pattern include the volume of page views and transactions, the volume and type of searches, the complexity of transactions, the volatility of the data, and security considerations.

IBM has identified five distinct workload patterns and corresponding Web site classifications.

### **Publish/subscribe Web sites provide users with information**

Sample publish/subscribe sites include search engines, media sites such as newspapers and magazines, and event sites such as those for the Olympics and for the championships at Wimbledon. Site content changes frequently, driving changes to page layouts. While search traffic is low volume, the number of unique items sought is high resulting in the largest number of page views of all site types. Security considerations are minor compared to other site types. Data volatility is low. This site type processes the fewest transactions and has little or no connection to any legacy systems.

### **Online shopping sites let users browse and buy**

Sample sites include typical retail sites where users buy books, clothes, and even cars. Site content can be relatively static, such as a parts catalog, or dynamic where items are frequently added and deleted as, for example, promotions and special discounts come and go. Search traffic is heavier than the publish/subscribe site, although the number of unique items sought is not as large. Data volatility is low. Transaction traffic is moderate to high and almost always grows. Typical daily volumes for many large retail customers who use IBM Net.Commerce range from less than one million hits per day to over 3 million hits per day. Of the total transactions, typically between 1% and 5% are buy transactions. When users buy, security requirements become significant and include privacy, non repudiation, integrity, authentication, and regulations. Shopping sites have more connections to legacy systems, such as fulfillment systems, than the publish/subscribe sites, but generally less than the other site types.

### **Customer self-service sites let users help themselves**

Sample sites include banking from home, tracking packages, and making travel arrangements. Data comes largely from legacy applications and often comes from multiple sources, thereby exposing data inconsistency. Security considerations are significant for home banking and purchasing travel services, but less so for other uses. Search traffic is low volume; transaction traffic is low to moderate, but growing.

### **Trading sites let users buy and sell**

Of all site types, trading sites have the most volatile content, the highest transaction volumes (with significant swing), the most complex transactions, and are extremely time sensitive. Trading sites are frequently tightly connected to the legacy systems, and nearly all transactions interact with the backend servers. Security considerations are high, equivalent to online shopping, with an even larger number of secure pages. Search traffic volume is low.



## Business-to-business sites let businesses buy from, sell to each other

Many businesses are implementing a Web site for their purchasing applications. Such purchasing activity may also be characteristic of other site types, such as publish/subscribe sites. Data comes largely from legacy applications and often comes from multiple sources, thereby exposing data consistency. Security requirements are equivalent to online shopping. Transaction volume is low to moderate, but growing. Transactions are typically complex, connecting multiple suppliers and distributors.

### 8.2.2 Scalability in practice

Meeting such challenges requires flexibility and capacity for change in all areas. The success of future Web sites may be tied to the selection of site components that can be individually or collectively adjusted to meet variable demands. This flexibility is called *scalability* and is a feature you need to understand and measure for each site component. Scalability is related to performance (response time) and capacity (operations per unit of time) but should not be considered synonymous.

Scaling a multitiered e-business solution from end to end means managing the performance and capacities of each component within each tier.

Figure 8-1 shows a sample architecture with the typical and major components. From a scalability viewpoint, the key components of an e-business solution are:

- ▶ The Web server
- ▶ The Web application server
- ▶ The network
- ▶ The directory and security servers
- ▶ The firewalls
- ▶ The existing business servers
- ▶ The database server

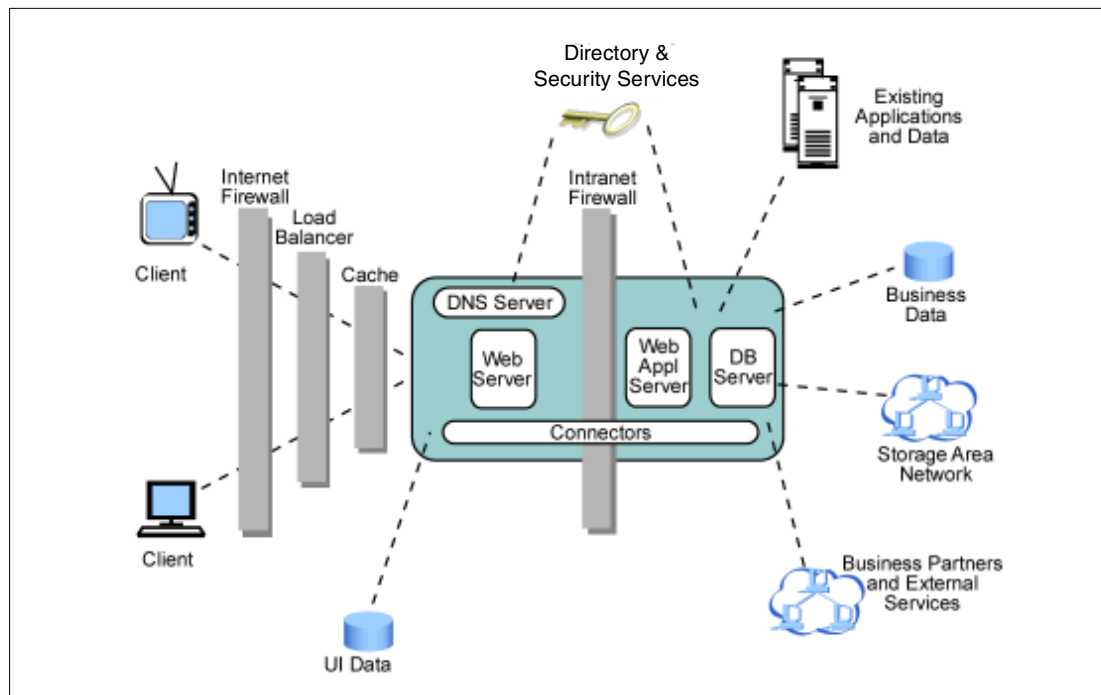


Figure 8-1 General architecture of a Web site

The basic objectives of scaling a component or system are to:

- ▶ Increase the capacity or speed of the component
- ▶ Improve the efficiency of the component/system
- ▶ Shift or reduce the load on the component

As you increase the scalability of one component, the result may change the dynamics of the site, thereby moving the bottleneck to another component. The scalability of the solution depends on the ability of each component to scale to meet increasing demands. Table 8-1 shows how the three principle scaling objectives may be met in terms of the various scaling techniques.

*Table 8-1 How scaling techniques relate to scaling objectives*

	Scaling technique	Increase capacity or speed	Improve efficiency	Shift or reduce load
1	Use a faster machine	X		
2	Create a machine cluster	X		
3	Use a special machine	X	X	
4	Segment the workload		X	X
5	Batch requests		X	
6	Aggregate user data		X	
7	Manage connections		X	
8	Cache data and requests		X	X

### 8.2.3 Six steps to scaling your solution

We now describe a high-level, systematic approach to classify your Web site and learning which scaling techniques could be applied. You will need to improvise and adapt the approach to your situation. The six steps are:

1. Understand the environment
2. Categorize your workload
3. Determine the components most impacted
4. Select the scaling techniques to apply
5. Apply the techniques
6. Re-evaluate

#### Step 1: Understand the environment

For existing environments, the first step is to identify all components and understand how they relate to each other. The most important task is to understand the requirements and flow of the existing applications and what can or cannot be altered. The application is key to the scalability of any solution, so a detailed understanding of the environment is mandatory to effectively scale any solution.

At a minimum, your analysis must include a breakdown of transaction types and volumes as well as a graphic view of the components in each tier.

As you analyze requirements for a new application, you have the opportunity to build scaling techniques into your solution. Each technique affects application design; similarly, application design impacts the effectiveness of the technique. To achieve proper scale, application design must consider potential scaling effects. In the absence of known workload patterns, you'll need to follow an iterative, incremental approach.

## Step 2: Categorize your workload

Your site may have high-volumes of dynamic transactions. Your site type will become clear as you evaluate your site for the other characteristics that pertain to transaction complexity, volume swings, data volatility, security, and others.

## Step 3: Determine the components most affected

This step involves mapping the most important site characteristics to each component. Once again, from a scalability viewpoint, the key components of an e-business solution are the Web server, the Web application server, the network, the directory and security servers, the firewalls, the existing business servers, and the database server.

## Step 4: Select the scaling techniques to apply to scale the workload

It is worth your best efforts to collect the information needed to make the best scaling decision. Only when the information gathering is as complete as possible is it time to consider matching scaling techniques to components. Here's a summary of the eight scaling techniques:

### ► Use a faster machine

This technique applies to the Web server, the Web application server, the network, the directory and security servers, the Internet and intranet firewalls, the existing business applications, and the database. The goal is to increase the ability to do more work in a unit of time by processing tasks more rapidly. A faster machine can be achieved by upgrading the hardware or software. However, one of the issues is that software capabilities can limit the hardware exploitation and vice versa. Another issue is that due to hardware or software changes, changes may be needed for existing system management policies.

### ► Create a cluster of machines

This technique applies to the Web server, the Web application server, the directory and security servers, and the intranet firewall. The primary goal here is to service more client requests. Parallelism in machine clusters typically leads to improvements in response time. Also, system availability is improved due to failover safety in replicas. The service running in a replica may have associated with it state information that must be preserved across client requests, and therefore, need to be shared among machines. State sharing is probably the most important issue with machine clusters and can complicate the deployment of this technique. IBM WebSphere uses an efficient data sharing technique to support clustering. Such issues as additional system management for hardware and software can also be challenging.

### ► Use a special machine

This technique applies to the Web server, the network, the directory and security servers, and the Internet and intranet firewalls. The goal is to improve the efficiency of a specific component by using a special purpose machine to perform the required action. These machines tend to be dedicated machines that are very fast and serve a specific role. Examples are network appliances and routers with cache, such as the IBM 2216 and IBM WebSphere Performance Pack. Some issues to consider regarding special machines are the sufficiency and stability of the functions and the potential benefits in relation to the added complexity and manageability challenges. Caching may reduce the HTTP server traffic significantly.

### ► Segment the workload

This technique applies to the Web server, the Web application server, the network, the intranet firewall and the database. The goal is to split up the workload into manageable chunks, thereby obtaining more consistent and predictable response time. The technique also makes it easier to manage on which servers the workload is being placed. Some of

the issues with this technique are that in order to implement the segmentation, you need to characterize the different workloads serviced by the component. After segmenting the workload, additional infrastructure is required to balance physical workload among the segments, for example, the use of the IBM network dispatcher (eND) technology.

► **Batch requests**

This technique applies to the Web server, the Web application server, the directory and security servers, the existing business applications, and the database. The goal is to reduce the number of requests sent between requesters and responders (such as between tiers or processes) by allowing the requester to define new requests that combine multiple requests. The benefits of this technique arise from the reduced load on the responders by eliminating overhead associated with multiple requests. It also reduces the latency experienced by the requester due to the elimination of the overhead costs with multiple requests. Issues may include limited ability to reuse requests due to inherent differences in various requests types (such as Web frontend and voice response frontend requests). Supporting different request types can lead to increased costs.

► **Aggregate user data**

This technique applies to the Web server, the Web application server, and the network. The goal is to allow rapid access to large customer data controlled by existing system applications and support personalization based on customer specific data. When accessing existing customer data spread across existing system applications, the existing applications may be overloaded, especially when the access is frequent. This can degrade response time. To alleviate this problem, the technique calls for aggregating customer data into a customer information service (CIS). A well-maintained CIS can provide rapid access to the customer data for a very large number of customers. Note that a CIS needs to scale very well to support large data as well as field requests from a large number of requesters.

► **Manage connections**

This technique applies to the Web server, the Web application server, and the database. The goal is to minimize the total number of connections needed for an end-to-end system, as well as to eliminate the overhead of setting up connections during normal operations. To reduce the overhead associated with establishing connections between each layer, a pool of pre-established connections is maintained and shared across multiple requests flowing between the layers. For example, most application servers provide database connections managers to allow connection reuse. It is important to note that a session may use multiple connections to accomplish its tasks or, many sessions may share the same connection. In the WebSphere connection manager, this is called *connection pooling*. The key issue with this technique is maintaining a session's identity when several sessions share a connection.

► **Cache**

This technique applies to the Web server, the Web application server, the network, the existing business applications, and the database. The goal is to improve the performance and scalability by reducing the path length traversed by a request and the resulting response, and by reducing the consumption of resources by components. Caching techniques can be applied to both static and dynamic Web pages. A powerful technique to improve performance of dynamic content is to asynchronously identify and generate Web pages that are affected by changes to the underlying data. Once these changed pages are generated, they must be effectively cached for subsequent data requests. Intelligent caching technologies can significantly enhance the scalability of e-business systems. The key issues with caching dynamic Web pages are determining what pages should be cached and when a cached page has become obsolete.

### Step 5: Apply the technique or techniques

Apply the selected technique or techniques in a test environment first to evaluate not only the performance or scalability impact to the component, but also the impact to its surrounding components. You do not want a situation where improvements in one component result in an increased (and unnoticed) load on another component. Figure 8-1 on page 171 illustrates the typical relationship between the techniques and the key solution components. By using this diagram, you can identify the key technique for each component.

Factors that could prevent you from applying one or more of the techniques may include:

- ▶ You currently cannot afford to invest in the technique, even if it would help.
- ▶ You determine that the technique will provide more scaling than you need.
- ▶ Your cost/benefit analysis shows that the technique will not result in a reasonable payback.

You must define a process for applying these techniques to yield the best return in different situations. Use this approach to map the starting points to focus on first based on your workload.

### Step 6: Re-evaluate

As with all performance-related work, tuning will be required. The goals are to eliminate bottlenecks, scale to a manageable status those that can't be eliminated, and work around those that can't be scaled. Here are some of the tuning actions that are used and the benefits that may be realized:

- ▶ Increasing the Web server threads raises the number of requests that can be processed in parallel.
- ▶ Adding indexes to the database server reduces I/O bottlenecks.
- ▶ Changing defaults for several operating system variables allows threaded applications to use more heap space.
- ▶ Caching significantly reduces the number of requests to the database server.
- ▶ Increasing the number of Web servers improves load balancing.
- ▶ Upgrading the database server increases throughput.

While scaling end-to-end e-business solutions remains more of an art than a science, this approach provides the a systematic way to respond to and manage unpredictable demands on your Web site. With thorough knowledge of your current workload patterns, Web site components, skills, and budget, you should begin now to factor scalability considerations and scaling techniques into the plan for the future.

## 8.3 Specific iSeries scaling considerations

The iSeries environment has several significant attributes that materially assist in scaling a WebSphere solution. Of most significance are the implementations of the JVM and DB2 UDB for iSeries, both of which are integrated into the operating system.

As the iSeries implements the concept of a Technology Independent Machine Interface (TIMI), it was possible to build the Java Virtual Machine into the OS/400 SLIC to provide a highly tuned, efficient, stable, and scalable implementation.

### 8.3.1 Overview

The iSeries offers many different advantages over other platforms for running WebSphere, including benefits from ease of operations to administration. The tools that you need to create your Web applications and debug them are all available to you. The total cost of ownership is proven lower for the iSeries. As a multi-user and multi-application platform, you won't need separate machines to run each piece of your Web application.

Specific iSeries advantages include:

- ▶ Integrated DB2 database
- ▶ Integrated JVM
- ▶ Access to legacy applications
- ▶ Coexistence Standard Edition and Advanced Edition
- ▶ Multiple instance support

The configuration customization may include:

- ▶ Creating a server without the Default Application Server
- ▶ Knowing when the Administrative Server is ready
- ▶ Administrative agent versus Administrative Server
- ▶ Configuring a remote Administrative Repository
- ▶ Creating a WebSphere cluster
- ▶ Configuring an Administrative agent
- ▶ Persistence session
- ▶ Configuring an HTTP server
- ▶ Using a Network Dispatcher
- ▶ Implementing a heterogeneous environment
- ▶ Using JDBC drivers

### 8.3.2 Integrated DB2 database

Because the iSeries has an integrated database, with a significant portion of the code executing within the SLIC, this may give rise to slightly different topology designs to those described in other operating system environments. Using a local database for an Administrative Repository provides faster accesses and more security than a remote database. As a standard, the OS/400 operating system and licensed products provide extensive tools that help manage, track, and maintain the database.

### 8.3.3 iSeries WebSphere scalability overview and key concepts

This section provides a conceptual overview of the goals and issues associated with scaling applications in the iSeries WebSphere environment.

#### Horizontal scaling considerations

If this instance is being created for use in a horizontal scaling topology, then you should use the same `<was_instance_root>` for all instances in this topology. This is recommended, because when you create models and clones, the `<was_instance_root>` directory used to qualify the standard out and error log files is that of the model's parent. If you use a different `<was_instance_root>` for instances in your horizontal scaling topology, then the log files will not be created due to an invalid directory structure.

## Other considerations

When you have multiple instances of the WebSphere Application Server Advanced Edition configured on a single iSeries server, you should ensure that the port information specified for the servlet engine does not conflict with other servlet engines from additional WebSphere Administrative Server instances. This is especially important if you are utilizing models and clones within your environment, because each successive clone increments this port number by one, starting from the port number originally specified. If you are running multiple instances of the WebSphere Application Server or you have multiple HTTP server instances accessing a single WebSphere Application Server instance, then you must ensure that you update the host alias information of your virtual hosts to reflect this.

## Persistent session information

If you will be utilizing persistent HTTP sessions in your environment and you plan to use horizontal scaling techniques for WebSphere workload management, you should create a persistent DataSource/JDBC driver using the Java Toolbox JDBC driver. Since only a single DataSource and JDBC driver are specified for all of the clones, it must work across all nodes in your environment.

If you are only using vertical scaling techniques, then you should use the iSeries Native JDBC driver.

## Network dispatcher environment

The existing WebSphere Performance Pack redbook, *IBM WebSphere Performance Pack: Load Balancing with IBM SecureWay Network Dispatcher*, SG24-5858, provides extensive information for configuring non-iSeries environments for use with the Network Dispatcher. This redbook talks about defining an alias to the cluster address on the loopback interface when configuring the Web server machines.

On the iSeries, you should equate this with creating an OS/400 TCP/IP virtual interface or virtual IP address for the cluster address. A virtual IP address can be thought of as a "System" IP address, which can only be contacted via an indirect route through a real TCP/IP interface. Since virtual IP addresses do not respond to ARP requests, the same virtual IP address can be defined on multiple machines, without causing any ARP conflicts, and allow several iSeries servers to appear as a single system to the outside world. When used in a Web environment, you would usually define a host entry for the virtual IP address. You would then configure your HTTP server to specifically bind to this host or virtual IP address.

### 8.3.4 Specific iSeries WebSphere scaling

To enable cloning, you need to create a regular application server and "promote" it to be a "model". Once the model is created, you can create a number of clones and locate them in any of the participating nodes. Cloning has two objectives: workload management and failover support.

There are various ways to distribute workload among the various clones. The WebSphere plug-in can route HTTP requests to any of the clones. Or you could use the IBM SecureWay Network Dispatcher if your topology included multiple separate boxes each running an HTTP server. Also, EJB workload management distributes method calls across a cluster of clones.

Workload can be transparently distributed across clones:

- ▶ By the WebSphere plug-in
- ▶ By the Network Dispatcher
- ▶ By EJBs Workload Management

WLM Transparently distributes EJBs across multiple application servers, which enables both horizontal and vertical scaling

## Horizontal scaling

Provides the ability to add more machines to the WebSphere Domain and maximizes utilization of machine resources.

Cloning is the key to WebSphere scalability. Horizontal cloning, when possible, is an obvious solution to increasing an environment's throughput. On other platforms, even on a single system, cloning can lead to better utilization of system resources by maximizing parallelism.

On the iSeries server, cloning is not necessary to scale well on a single system. A well-written application shouldn't require cloning to get maximum performance on the iSeries server. Typically, attempting to use cloning on a single system to improve performance actually hurts performance instead. This is because cloning requires the use of persistent sessions, which can cause a degradation in performance. By adding clones, you can generally "buy back" the throughput that you lost by turning on persistent sessions. But, in most cases you won't end up significantly ahead of where you started without clones.

In any case, make sure that you tune your application and iSeries server before you clone. It is obviously easier to tune a single application server and then clone, than trying to tune each individual clone afterwards.

## Vertical scaling

The main motivation behind attempting vertical scaling is to achieve a better use of the multiprocessing facilities provided by an operating system. An operating system typically does a better job at maximizing parallelism (SMP, CPU and I/O parallelism, multithreading, etc.) than the JVM itself.

If you have a very large number of clients (concurrent threads in an application server) and you run on a multi-processor system, for example, it may be a good idea to experiment with cloning in order to possibly achieve higher throughput.

However, keep in mind that even on an n-way system, the rule of thumb that suggests one Application Server per CPU may actually not necessarily hold true, especially with WebSphere Application Server 3.5. It is possible for a single application server to fully and efficiently utilize the CPUs.

Vertical scaling includes the following aspects:

- ▶ Garbage collection still a problem after tuning
- ▶ Fewer clients per application server may lead to better garbage collection
- ▶ Fix any memory leak problem first
- ▶ Maximize parallelism on an n-way machine
- ▶ Take advantage of the OS superior multithreading
- ▶ Minimize contention on resources that are duplicated with every JVM clone
- ▶ Low-level Java semaphores, Java monitors, number of DB connection per JVM



## Cloning an application server

**Important:** Before you start cloning, the first and most important rule is to tune the application server.

Cloning has two objectives: workload management and failover support. Cloning is the key to WebSphere scalability. Horizontal cloning, when possible, is an obvious solution to increasing an environment's throughput. Also, if you run on a multiprocessor machine, it is worthwhile to try cloning to attempt to achieve better throughput.

### EJB workload management

A Java client issues a method call on an EJB remote reference. In WebSphere, you can "WLM-enable" the EJB stubs. This process will cause the stub to become aware of the existence of replicas of the enterprise beans that may exist on separate application servers (typically clones). Figure 8-2 shows the concept of EJB workload management over multiple Application Server clones.

From the client perspective, there is no difference in calling a method on a WLM-enabled bean or on a normal bean. The WLM-enabled stubs, however, will dynamically route the method call to any one of the available cloned instances of the bean. There are, of course, rules that regulate how these methods are routed. Typically, during an individual transaction, the stubs tend to direct a client's method calls to the same application server (transaction affinity).

In addition, if the call occurs from within the same JVM as the bean resides, WLM is bypassed and the call occurs locally. This is the case of a servlet that calls a method on a bean that runs in the same JVM.

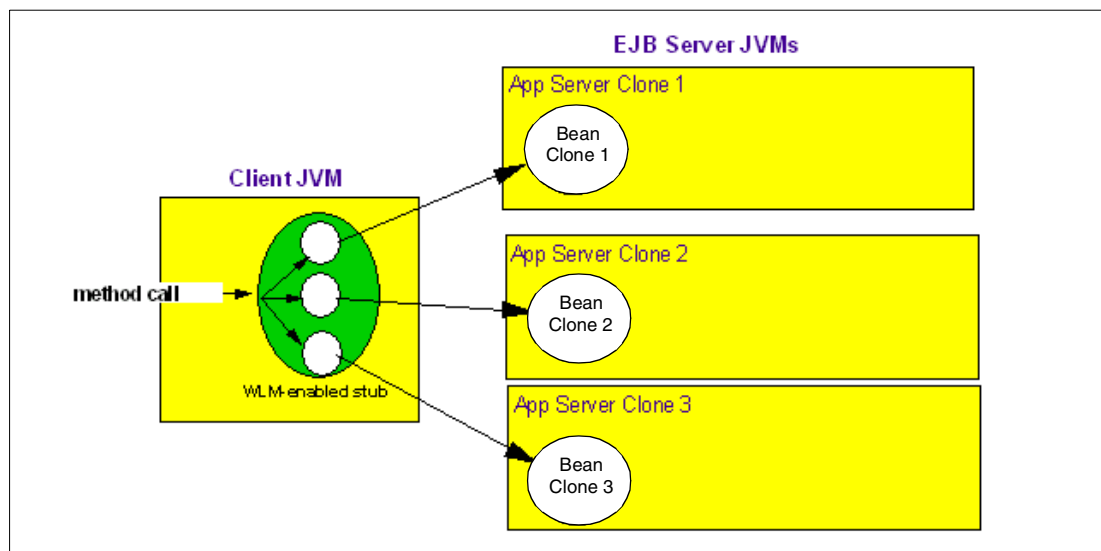


Figure 8-2 EJB workload management over multiple application servers

An important exception is represented by stateful session beans. Stateless session beans can be easily distributed across multiple servers. They are not client specific; they don't contain any externally accessible state; in other words, they contain pure "logic". Entity beans, on the other hand, can be cloned too, because each of the clones can retrieve the state from the RDB. But stateful session beans do not have a representation on the database and are client specific. A stateful session bean only makes sense to a specific client. WebSphere doesn't support WLM for the stateful session beans (however, their homes are WLM-enabled).

## Servlet clustering

Servlet clustering is one of the principal keys to scalability. We suggest that keeping servlets and EJB in the same JVM does not require WLM to play a role. Using session beans as a facade is still a good practice, since it shields servlet code from business object model changes.

Another major issue is keeping track of context information, for example, a shopping cart. The use of stateful session beans is hazardous, since it could lead to disastrous performance in some cases. We suggest that the context should be maintained as attributes of the HTTP session, by the use of persistent HTTP sessions and stateless session beans. You should attempt to minimize amount of data stored in the HTTP session. Persistent sessions need to be used to be shared by the multiple clones. The architecture then is more fault tolerant, but still remains client-specific.

In this case, scalability is ensured by "servlet clustering". The HTTP requests will be distributed across a number of available clones. WebSphere implements session affinity that ensures that, after a while, a client keeps connecting to the same server, unless this server is down.

Keeping servlets and EJB in the same server is good practice in this scenario, because it reduces I/O traffic dramatically. Using session beans is still good practice in this case, but stateful session beans can lead to disastrous performance. If a client creates a stateful session bean on server A and the next request ends up being processed on server B, the session bean will still be accessed on Server A, with a strong performance premium being paid.

### 8.3.5 Current iSeries JDBC driver limitations

When using JDBC to access the iSeries database, two JDBC drivers are available. They are the IBM Toolbox for Java JDBC driver and the Native JDBC driver. When running on the iSeries server, for example under WebSphere Application Server, you should use the Native JDBC driver. It performs better. Be aware of the following limitations of the IBM JDBC drivers.

#### Two-phase commit capabilities

Due to limitations in the iSeries host servers, two-phase commit is not available when using the Java Toolbox JDBC driver. If you require two-phase commit capabilities, then you must use the Native iSeries JDBC driver.

#### Accessing remote databases with native JDBC driver

Currently, DRDA is not fully implemented over TCP/IP in the OS/400 operating system. When you require remote iSeries database access, especially when you require two-phase commit or use large objects, such as BLOB or CLOBs, you must configure your system with the Native JDBC driver to use DRDA over SNA.

**Note:** For the latest information on how to set up the iSeries Toolbox driver and the iSeries Native Driver for remote Administrative Repository, please refer to the latest iSeries WebSphere documentation and addendum. You can find Web links to the documents in "Referenced Web sites" on page 213.



## Sizing and capacity planning

This chapter looks at some of the more important aspects of both sizing and capacity planning for the iSeries WebSphere Application Server environment. First it discusses a generic capacity management process, applicable to all environments, and then it considers specific aspects as they apply to our target environment.

## 9.1 Why do performance and capacity planning

How much should you invest in managing the performance of your system? The needs of your business change, sometimes sooner than you expect. To respond to business changes effectively, your system must change too. Managing your system, at first glance, might seem like just another time-consuming job. But the investment pays off soon because the system runs more efficiently, and this is reflected in your business. It is also efficient because changes are planned and managed.

Sometimes good performance just happens. In those cases, the system has plenty of resources to get the job done, but there are times when those resources are not in the right place. Maybe systems and clients have been added to the network, or production volume has increased and the workload is significantly changed. Or, more often, workload changes in small, nearly invisible increments, and one day, performance is just not as good anymore. That is why you have to plan ahead for your system to be at peak performance, especially in a quick-paced business.

As a result, it is important to manage performance effectively for both the long term and the short term. In the short term, understanding the performance components of your system helps you react quickly when a performance problem occurs at a crucial time. It may also allow you to defer upgrading for a few months. In the long term, if you plan for a more efficient system, you prevent potential performance problems from developing. You also ensure that you have enough capacity on the system to handle your workloads. In addition, your users get the service they expect. Maintaining good performance requires you to understand, plan, and manage performance.

Effectively managing performance means developing your own performance strategy. Performance management is necessary to optimize the use of a system and its associated services. Performance management is a strategy for planning, implementing, controlling, and measuring computer-based tasks to achieve performance that is acceptable.

You should also be aware that some results do not happen immediately after a change. There is a delay or lag before the effect is seen. This is indicated in Figure 9-1, which shows a composite cause-effect graph.

When tracing the path of one loop, the following actions occur:

- ▶ When new users are added to the system, after a delay, usage will tend to rise.
  - As a result of the increased usage, there will be a gradual (delayed) increase in resource contention.
  - As a direct result of the increased resource contention, there is likely to be an (immediate) increase in effective response, small though this may be.
  - Increased response has a delayed tendency to reduce usage.
- ▶ This loop exhibits “negative feedback” characteristics in that there is a self-limiting effect.

It should be noted that in a diagram of this type, all loops must have this negative feedback effect, as shown by an arrow with a plus sign (+) at one end and a minus sign (-) at the other end.

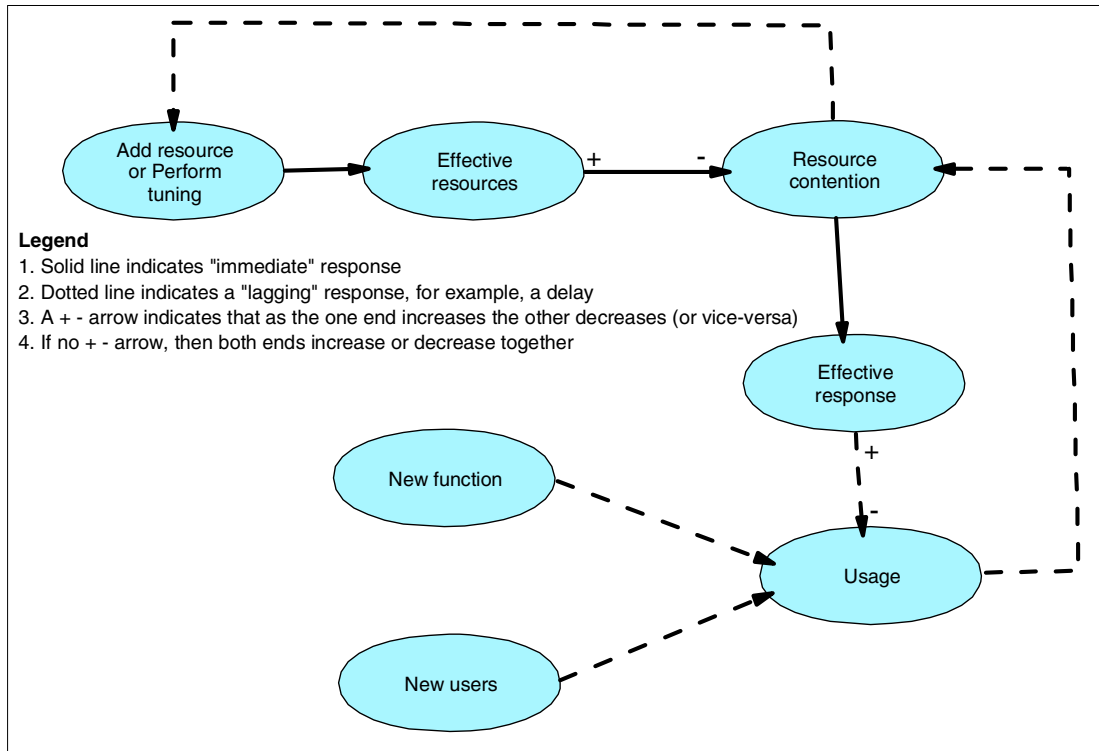


Figure 9-1 Cause-effect graph

## 9.2 Generic performance process and methodology

This section reviews an overall process for performance monitoring and capacity planning, which is shown in Figure 9-2. The perspective is generally from that of a performance specialist.

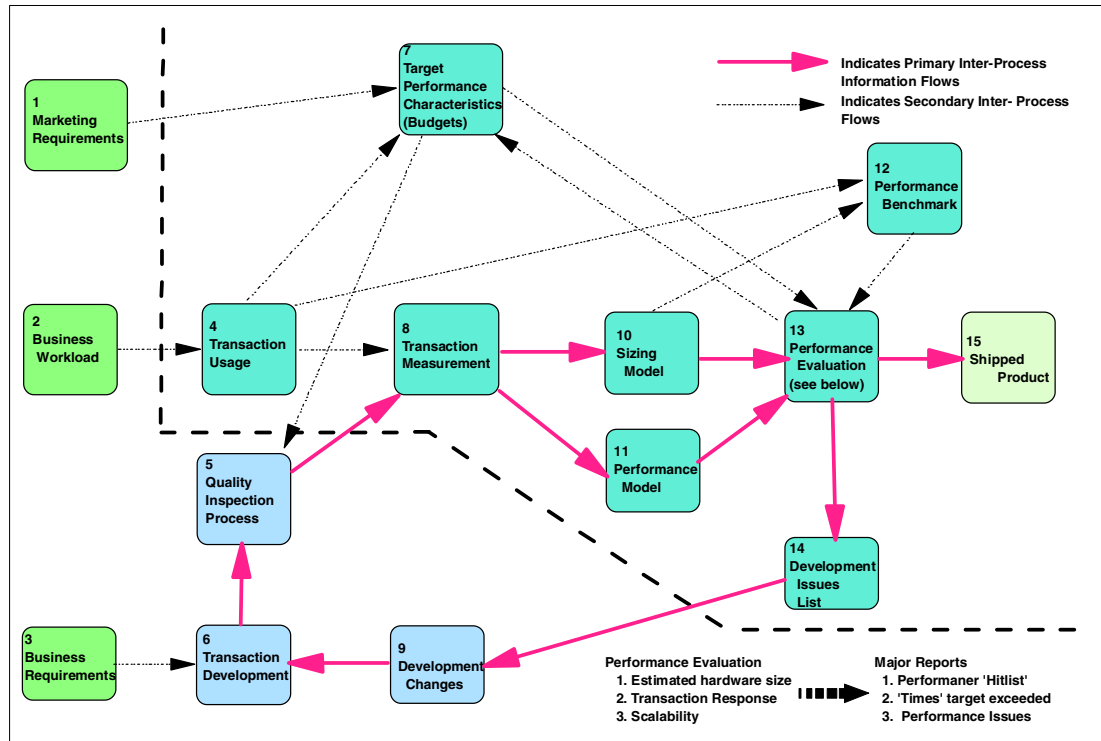


Figure 9-2 Generic performance process

All developers need to be aware of the process, even though they may not want or need to be more familiar with the detailed measuring aspects. The need for specific tuning work on an individual application/program is most likely to be generated from the performance architect or specialist. But, this does not absolve developers from being aware of the performance aspects of their design and coding. Ideally, provided that the applications have been well designed and implemented in a performance-conscious manner, there should be little need for significant tuning effort. These notes outline the performance methodology successfully used in a development organization. Overall the various process elements can be categorized in a number of ways:

- **Macro:** At the highest level of granularity, either a business transaction or a machine transaction.
- **Micro:** At the machine transaction executable statement level.

Significantly there is an essential feedback process with the following steps:

1. Set resource (performance) requirements
2. Measurement
3. Evaluation against targets and assessment of the impact of changes
4. Plan changes
5. Implement changes and, if required, re-set requirements

## 9.2.1 Process steps

Table 9-1 describes in greater detail the more important process steps from Figure 9-2.

Table 9-1 Detailed description of process steps

Step	Title	Description
4	Analyses Transaction Usage (or Flows)	<p>Starting from a defined Business Workload, termed the “Reference Workload”, map to the individual machine transactions that implement the business transactions.</p> <p>Accumulate the occurrence frequencies of each machine transaction, which determines the relative importance of each transaction in the server workload. From this analysis, the “key” transactions are established, those which in total comprise 80+% of the workload.</p> <p>The information also is a primary input to the Sizing model.</p>
7	Target Performance Characteristics	<p>Set Development budgets based on primary input documents that may include:</p> <ul style="list-style-type: none"> <li>▶ Non-functional requirements</li> <li>▶ Defined physical processes</li> <li>▶ Business Transaction to Machine Transaction flows</li> <li>▶ Tests and measurements carried out as part of preliminary analysis</li> </ul>
8	Measure Server Transaction Characteristics	<p>Using server monitoring tools as described elsewhere in this document, measure machine transaction resource use. Based on these measurements the updates the Sizing model is updated to provide a “how-goes-it” evaluation.</p>
10 & 11	Sizing and Performance Models	<p>Models that provide an assessment, based on the reference workload plus transaction measurements, of how closely the developed transactions in total meet the required performance levels.</p>
12	Performance Benchmark	<p>This provides the measure of how a simulated live load will execute in a defined environment. Interim mini-benchmarks and stress testing runs provide early indications prior to a fully functional benchmark.</p>
13	Performance Evaluation (Evaluate Server Transaction Characteristics)	<p>Based on sizing targets, evaluate the transactions and determine the need for specific transactions to improve performance. Both design and implementation are considered. Detailed measurements (at the executable statement level) may be required in selected circumstances. The Measurement and Evaluation processes result in updates to the sizing model, and re-calculation of the current status of the sizing against the target. Management feedback is extracted.</p>
14 & 9	Define Improvement Strategy/Strategies	<p>Give, to the development team or teams, the measurement results and suggested transaction changes that need to be implemented to close in on the target figures.</p>

It is a fundamental precept of performance evaluation and tuning that *the effort is placed where maximum gain will be achieved*. Therefore, the specific tuning must be considered in the context of the whole application, not just an individual component.

Given a set of measurements, the granularity dependent upon the tool used, then the values must be compared to established targets, and an assessment made of the improvements possible by making changes. There must be a balance between the effort required to measure, and the likely improvement to be gained. For example, key transactions that comprise a significant component of the server workload are obviously worth spending considerable effort to improve, while a transaction that executes once per hour or once per day probably does not deserve the same effort. Similarly tuning common code, such as frameworks that will apply to all transactions, is a significant activity.

The basic questions that you must consider are:

- ▶ How frequently is this transaction executed?
- ▶ What is the contribution to total hardware sizing? What resources does the transaction consume?
- ▶ What improvement can be gained by change? And what is the impact of that change on work completed?
- ▶ Are there any common factors that apply to all transactions, indicating that common code could or should be changed?

Changes may apply to:

- ▶ *Design*: In which case a rewrite may well be required
- ▶ *Reuse*: Whether appropriate, in which case implementation needs to be modified
- ▶ *Code*: Again implementation needs to be modified

Figure 9-3 shows the performance activity profile, which indicates how performance resources may be required during the life of a development project.

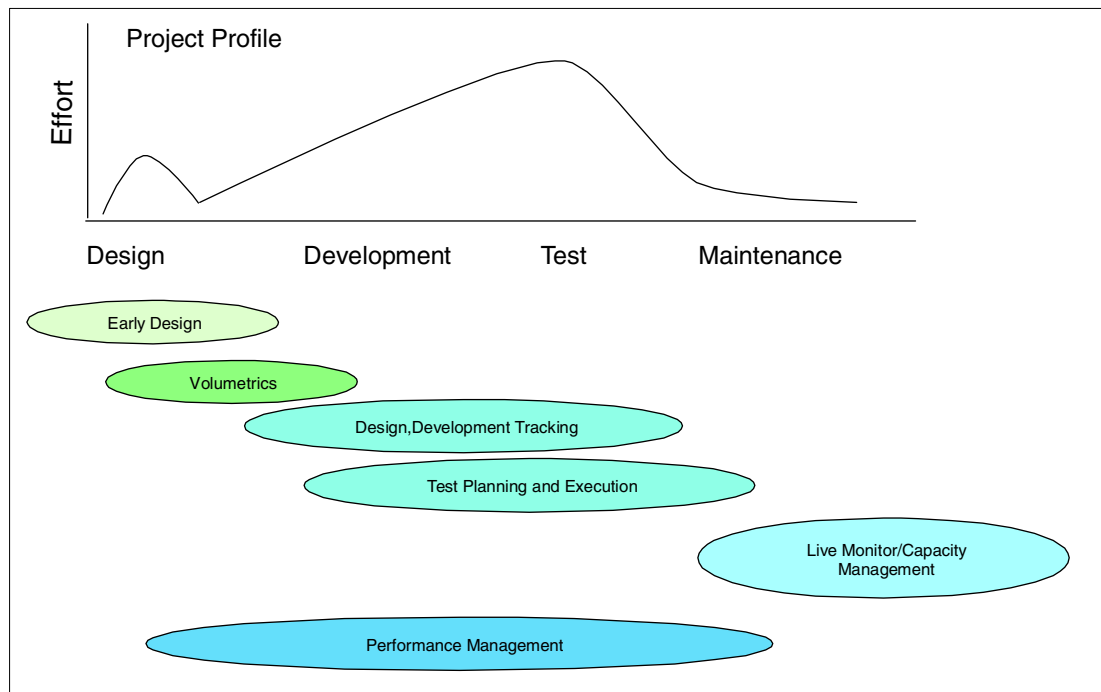


Figure 9-3 Performance activity profile

## 9.3 Capacity planning and workload estimation tools

There are no easy answers to capacity planning questions. Use the IBM Workload Estimator for iSeries to size an iSeries server for WebSphere. You should also refer to *iSeries Performance Capabilities Reference Version 5 Release 1*. Consult the following chapters in this document:

- ▶ Chapter 22 for information on the Workload Estimator
- ▶ Chapter 6 for information on Webserving performance capacity planning
- ▶ Chapter 7 for general Java performance information



“Referenced Web sites” on page 213 provides the Web link to the Performance Capabilities Reference and Workload Estimator.

The deployment environment is rich in choices. The developers on iSeries who want to deploy a WebSphere solution can make many trade-offs between using WebSphere function (shortening development) and “rolling their own” code (improving performance). There is nothing better than some sort of working prototype, even a partial prototype, for making estimates. Such information, if it is available, should be used in lieu of standard planning tools.

Use Performance Management (PM/400) and the BEST/1 capacity planning tools to plan for future resource demands on system performance. PM/400 allows you to easily submit your utilization and growth data to the Workload Estimator. Use the Workload Estimator to size your next upgrade.

### 9.3.1 BEST/1

BEST/1 is the iSeries tool provided to:

- ▶ Model current system performance
- ▶ Predict hardware upgrades needed to meet current performance objectives
- ▶ Predict future hardware needs based on business growth projections
- ▶ Predict performance of additional or changed applications

Capacity planning is an ongoing process used to determine current system performance and future data processing needs. In its simplest form, using BEST/1 involves a five-step process:

1. Collect the performance data with the Collection Services part of Operations Navigator.
2. Create the performance data from the data collection with the CRTPFRDTA command.
3. Create a model from the performance data.
4. Analyze the BEST/1 model.
5. Perform a “What-if...?” analysis and continue the analysis process until you are satisfied with the results.

BEST/1 is an analytic modeling tool that provides expert evaluations and configurations. As such, the results depend entirely on the accuracy of the input, the analytic model, and the rule base for evaluations and configurations.

As with most capacity planning, there are some conditions or assumptions present. The following list represents the major assumptions under which BEST/1 performs its modeling work:

- ▶ Data is collected on a well-tuned, stable system.
- ▶ Data collection is done on a system in which all pool sizes and activity levels are set to reasonable values, secondary paging (thrashing) does not occur, and if excess storage is present, working set sizes are appropriate for the activity level of the pool.
- ▶ When data collection is done, the time slices are large enough to minimize active-to-ineligible transitions, which are shown as “A-I” in the Performance Tools Transaction Report Interactive Jobs output. These are also visible on the transition data shown in WRKSYSSTS display.
- ▶ The workload is reasonably homogeneous. That is, one particular program within a workload is not causing a resource limitation.
- ▶ The workload is in steady-state. This means that the number of interactive users and batch tasks is relatively constant.

- ▶ The proportion of a workload's I/Os to an auxiliary storage pool (ASP) is based on the proportion of total I/Os handled by the disks on that ASP. Read and write operations are distributed on the same basis. There are no severe resource limitations (bottlenecks) in the system. That is, none of the primary devices (processing unit, disk, and memory) are excessively utilized.
- ▶ All jobs specified as active are performing work.
- ▶ The average service time of each disk request is the same. All disk requests are spread evenly across all disk drives in each ASP.
- ▶ The activity on the local work station IOPs is spread evenly across all IOPs, and the utilization of each IOP is the same. The service time and response time depend on the application.
- ▶ All controllers are equally distributed across all communications lines for LAN and WAN.
- ▶ All local, LAN, and WAN controllers have the same service time, respectively.

As the system approaches the saturation point (high utilization) of one or more resources, the accuracy of the analytic model for response time prediction diminishes. An analytic model is best used to predict when a resource will become saturated. Knowing when a resource will become saturated allows you to plan your workloads and hardware upgrades beforehand.

The steps to follow during the modeling process are:

1. Collect the data as shown in 3.4.1, "Collecting performance data" on page 19.
2. Create performance data as shown in 3.4.2, "Creating performance data" on page 22.
3. Create a model.
4. Analyze the model.
5. Experiment with the workload.
  - Growth in workload
  - Different configurations (CPU/memory)

### **Example of creating a model**

You should have at least one collection of performance data available before you start to create a model. Because we had multiple data members to choose from, we chose the one that represents a relatively heavy workload. We follow these steps:

1. Enter the Start BEST/1 (STRBEST) command. The BEST/1 for the AS/400 menu appeared as shown in Figure 9-4.
2. Choose option 1 to work with BEST/1 models. The reason for choosing this option is that it is the only option that enables us to classify jobs to workloads, which is shown in Figure 9-8 on page 191. Press Enter.

```

BEST/1 for the AS/400

Select one of the following:

    1. Work with BEST/1 models

    5. Create BEST/1 model from performance data

    10. Work with results

    50. General information and tutorial

    60. More BEST/1 options

Selection or command
===> 1

F3=Exit  F4=Prompt  F9=Retrieve  F12=Cancel  F21=Advanced user level

```

Figure 9-4 Creating a model

3. You see the screen shown in Figure 9-5. Enter a name and a meaningful description for the model you are creating, because you cannot change the name or description after the model is generated. Press Enter.

```

Work with BEST/1 Models

Library . . . . . QGPL      Name

Type options, press Enter.
  1=Create  3=Copy  4=Delete  5=Work with  6=Print  7=Rename

Opt Model      Text                                     Date      Time
  1 YOURMODEL  GIVE A MEANINGFUL NAME
    PETRIDIDIT                                     10/26/01  11:15:21
    SAFEMODEL  Some descriptive text here          10/25/01  17:26:07
    ACOWASHERE And did this                        10/25/01  17:06:44
    BASEMODEL  A really descriptive text here      10/25/01  14:23:54

Command
===>

F3=Exit  F4=Prompt  F5=Refresh      F9=Retrieve  F12=Cancel
F15=Sort by model  F16=Sort by text  F19=Sort by date and time

```

Figure 9-5 Naming a model

4. This leads you to the screen shown in Figure 9-6. Specify the library, name, and the criteria of selecting the performance data from which you will create the model.

```

                                Create BEST/1 Model from Performance Data

Model . . . . . : YOURMODEL

Type choices, press Enter. Use *SLTHOUR to select an hour-long time period or
use *SLTITV to select select first and last interval of a one to two hour
time period. The time period selected should be representative of your peak
processing activity.

Text . . . . . A DESCRIPTIVE TEXT HERE

Performance data:
  Member . . . . . TAKEONE      Name, F4 for list
  Library . . . . . FROMLIB     Name

Start time . . . . . *SLTITV    Time, *FIRST, *SLTHOUR, *SLTITV
Start date . . . . . *FIRST     Date, *FIRST

Stop time . . . . . *LAST       Time, *LAST
Stop date . . . . . *LAST       Date, *LAST

F3=Exit  F4=Prompt  F12=Cancel

```

Figure 9-6 Selecting the library and member

- On the screen shown in Figure 9-7, specify the first and the last interval to include in the model. You should choose data that contains a busy system, but not data where the CPU is saturated. If, for example, a looping application program is causing a CPU utilization of 99% and you build a model from that data, the model does not reflect the real workload.

```

                                Select Time Interval

Library . . . . . : FROMLIB      Performance member . . : TAKEONE

Type option, press Enter. Select first and last interval.
1=Select

Opt  Date      Time      ----Transaction----  --CPU Util--  I/Os per Sec
      Date      Time      Count  Rsp Time  Total  Inter  Sync  Async
1    10/25/01   09:15:00    104     .0      2      0      9      2
      10/25/01   09:20:00     69     .0      6      0     57     18
      10/25/01   09:25:00      0     .0     51      0    181    175
1    10/25/01   09:30:00      0     .0     91      0    339    300
      10/25/01   09:35:00      0     .0     99      0    277    202
      10/25/01   09:40:00      2     .0     90      0    360    248
      10/25/01   09:45:00     62     .0     94      0    344    234
      10/25/01   09:50:01      0     .0     99      0    274    197
      10/25/01   09:55:00      0     .0     75      0    295    207
1    10/25/01   10:00:00      0     .0     95      0    346    223
      10/25/01   10:05:00      5     .1      4      0     10      8
                                          More...

F3=Exit  F12=Cancel  F15=Sort by interval  F16=Sort by count
F17=Sort by rsp time  F18=Sort by total CPU util  F19=Sort by total I/Os

```

Figure 9-7 Selecting the time interval

6. While collecting the performance data (see 3.4.1, “Collecting performance data” on page 19, on how to collect performance data), independently kept track of the amount of orders generated by OrderEntry application, so you know exactly how many orders were generated between 09:30 and 10:00. You need the number of actual orders generated when predicting the effects of workload growth that is shown in Figure 9-23 on page 200. In this case, we know that we generated 800 orders in 30 minutes, which equals 1600 orders per hour.
7. On the screen shown in Figure 9-8, start the process of creating a model that only includes the data you will be interested in when you start increasing the workload. The actual modeling is shown in “Modeling the workload growth” on page 195. If you cannot access the screen shown in Figure 9-8, start the model creation again and make sure you start as shown in Figure 9-4 on page 189.

Classify Jobs

Select one of the following:

1. Use default job classification
- 2. Classify jobs into workloads**
3. Use existing job classifications

Selection

**2**

F3=Exit    F12=Cancel

*Figure 9-8 Classifying jobs into workloads*

8. On the screen shown in Figure 9-9, specify that you will classify jobs to a workload on a subsystem basis.

Specify Job Classification Category

Type choice, press Enter.

Category . . . . . 6

1=User ID

2=Job type

3=Job name

4=Account code

5=Job number

6=Subsystem

7=Pool

8=Control unit

9=Comm line

10=Functional area

F3=Exit

F12=Cancel

Figure 9-9 Classifying jobs by subsystem

9. Choose option 6, which takes you to the screen shown in Figure 9-10.
10. Press F9 to receive the values from the performance data. Pressing the F9 on the Edit Job Classifications display gives you the ability to assign jobs to workloads. This is shown in Figure 9-11.

Edit Job Classifications

Enter workload names and category values which are assigned to each workload, press Enter. Jobs with unassigned values become part of workload QDEFAULT.

Workload	Subsystem	Workload	Subsystem	Workload	Subsystem
_____	_____	_____	_____	_____	_____
_____	_____	_____	_____	_____	_____
_____	_____	_____	_____	_____	_____
_____	_____	_____	_____	_____	_____
_____	_____	_____	_____	_____	_____

F3=Exit

F9=Display values from data

F12=Cancel

To display values from performance data, press F9

More...

Figure 9-10 Getting values from the measured data

11. On the screen in Figure 9-11, include the work performed in subsystems QEJBSBS and QSYSWRK into the workload named WASANDJAVA. Pick up the work that is not done in

a subsystem that represents the operating system and licensed internal code programs (shown as a subsystem with no name). Press Enter.

Assign Jobs to Workloads

Workload . . . . . WASANDJAVA

Type options, press Enter. Unassigned jobs become part of workload QDEFAULT.  
1=Assign to above workload 2=Unassign

Opt	Workload	Subsystem	Number of Transactions	CPU Seconds	I/O Count
1			0	12.882	14698
		QBATCH	0	.478	215
		QCMN	0	.000	0
1		QEJBSBS	0	2912.844	8757
		QHTTPSVR	0	16.507	3
		QINTER	64	.396	257
		QSERVER	0	.002	6
1		QSYSWRK	0	872.728	1146196
		QUSRWRK	0	.000	0

Bottom

F3=Exit F12=Cancel F15=Sort by workload F16=Sort by subsystem  
F17=Sort by transactions F18=Sort by CPU seconds F19=Sort by I/O count

Figure 9-11 Assigning subsystems to a workload

12.The screen shown in Figure 9-12 appears. Accept the default values on this screen.

Specify Paging Behaviors

Type choices, press Enter.

Workload	Paging Behavior (F4 for list)
QDEFAULT	*GENERIC
WASANDJAVA	*GENERIC

Bottom

F3=Exit F4=Prompt F12=Cancel

Figure 9-12 Specifying the paging behavior

13. On the screen shown in Figure 9-13, specify the metrics used to represent non-interactive transactions. Because there are no hooks in the application telling the transaction start and end boundaries, you can assume that the amount of logical I/O generated equals the 800 transactions that we know took place when collecting the performance data.

```

                                Define Non-Interactive Transactions

Job classification category . . . . . :   Subsystem

Type choices, press Enter.

      ---Activity Counted as Transaction---
Workload      Type      Quantity      Total Transactions
QDEFAULT      *LGLIO      100.0          when Type = *NONE
WASANDJAVA     *LGLIO      100.0          0
                                                    0

                                                    Bottom

Type:  *LGLIO, *CMNIO, *CPUSEC, *PRINT, *NONE

F3=Exit  F12=Cancel

```

Figure 9-13 Defining a non-interactive transaction

14. After you define a non-interactive transaction, press Enter. You see the screen shown in Figure 9-14 that enables you to save the workload member.

```

                                Save Job Classification Member

Change values if desired, press Enter.

Member . . . . . WASANDJAVA      Name
Library . . . . . QGPL           Name

Text . . . . . JAVA, WebSphere and System work load

Replace . . . . . Y              Y=Yes, N=No

F12=Cancel

```

Figure 9-14 Saving the job classification member



15. After you save the workload member, you see Confirm Creation of BEST/1 Model screen (Figure 9-15). The actual creation of the model is done in a batch job that is submitted on the same display shown in Figure 9-15.

Confirm Creation of BEST/1 Model

Type choices, press Enter.

Model . . . . .	YOURMODEL	Name
Library . . . . .	QGPL	Name
Text . . . . .	A descriptive text here	
Replace . . . . .	N	Y=Yes, N=No
Job name . . . . .	CRTBESTMDL	Name, *JOB
Job description . . . . .	QPFRJOB	Name, *NONE, *USRPRF
Library . . . . .	QPFR	Name, *LIBL, *CURLIB

F12=Cancel

Member WASANDJAVA has been saved

Figure 9-15 Submitting the creation of model

Modeling the workload growth

Use the BEST/1 tool to predict how your existing hardware would perform if the workload would grow. You can also use the BEST/1 to simulate how the same workload would run on different hardware. In both cases, the steps to be taken are the same:

- 1. Collect performance data
- 2. Create performance data
- 3. Create a model
- 4. Use the model to simulate the changes

To model the workload growth, we follow these steps:

- 1. Since we already created the model shown in “Example of creating a model” on page 188, enter the STRBEST command and choose option 1 to work with models. This leads you to the screen shown in Figure 9-16.
- 2. Choose option 5 to work with a specific model and press enter. This causes the BEST/1 software to read the model into its workspace.

```

                                Work with BEST/1 Models

Library . . . . . QGPL          Name

Type options, press Enter.
  1=Create  3=Copy  4=Delete  5=Work with  6=Print  7=Rename

Opt Model      Text                                     Date      Time
  5  YOURMODEL  A descriptive text here                 10/25/01  14:23:54

                                                                    Bottom

Command
===>
F3=Exit  F4=Prompt  F5=Refresh          F9=Retrieve  F12=Cancel
F15=Sort by model  F16=Sort by text  F19=Sort by date and time

```

Figure 9-16 Starting to work with a model

3. After the model is read, the screen shown in Figure 9-17 appears. Choose option 5 to analyze the model.

```

                                Work with BEST/1 Model

Performance data . . . : QMPGDATA (TAKEONE)
Model/Text . . . . . : YOURMODEL  A descriptive text here

Select one of the following:

    1. Work with workloads
    2. Specify objectives and active jobs

    5. Analyze current model
    6. Analyze current model and give recommendations
    7. Specify workload growth and analyze model

    10. Configuration menu
    11. Work with results

                                                                    More...

Selection or command
===> 5
F3=Exit  F4=Prompt  F9=Retrieve  F12=Cancel  F15=Save current model
F17=Analyze using ANZBESTMDL  F22=Calibrate model  F24=More keys
Model YOURMODEL has been read

```

Figure 9-17 Starting to analyze a model

4. This leads you to the Work with Results screen shown in Figure 9-18. Choose option 5 to display the Analysis Summary.

```

                                Work with Results

Printed report text . . . . . A descriptive text here


Type options, press Enter.
  5=Display  6=Print

Opt   Report Name
      Measured and Predicted Comparison
5    Analysis Summary
      Recommendations
      Workload Report
      ASP and Disk Arm Report
      Disk Resources Report
      Main Storage Pool Report
      Communications Resources Report
      All of the above


                                                                    Bottom
F3=Exit  F12=Cancel  F14=Select saved results  F15=Save current results
F18=Graph current results  F19=Append saved results  F24=More keys
Model has been analyzed

```

Figure 9-18 Work with Results screen

5. This leads you to the Display Analysis Summary screen shown in Figure 9-19. Verify that the model resembles the load that was on the system during the time the performance data was collected. Press the F11 key to compare the measured data with the model you created.

```

                                Display Analysis Summary

CPU Model / release level . . . . . : 23F5          V5R1M0
Main Storage . . . . . : 4096          MB

                                Quantity    Predicted Util
CPU . . . . . : 2/2          92.4
Database . . . . . :          92.4
Disk IOPs . . . . . : 0          .0
Disk ctls . . . . . : 0          .0
Disk arms . . . . . : 12        3.3

                                                                    More...
                                Interactive    Non-interactive
CPU utilization % . . . . . : .0          92.4
Transactions per Hour . . . . . : 110        58135
Local response time (seconds) . . . . . : .0          .8
LAN response time (seconds) . . . . . : .5          2.0
WAN response time (seconds) . . . . . : .0          .0

Performance estimates -- Press help to see disclaimer.
F3=Exit  F10=Re-analyze  F11=Measured and predicted comparison  F12=Cancel
F15=Configuration menu  F17=Analyze multiple points  F24=More keys

```

Figure 9-19 Display Analysis Summary

6. An example of this is shown in Figure 9-20. If the Measured values are close to the Predicted values, assume that the model is usable for your purposes. If the values differ more than 10%, we recreate the model. After you verify the model, press Enter.

Measured and Predicted Comparison		
	Measured	Predicted
Total CPU util . . . . . :	92.4	92.4
Database util . . . . . :	92.4	92.4
Disk IOP util . . . . . :	.0	.0
Disk arm util . . . . . :	3.2	3.3
Disk IOs per second . . . . . :	569.1	564.8
Multifunction IOP util . . . . . :	2.3	2.4
Disk IOA util . . . . . :	8.6	8.6
LAN IOA util . . . . . :		.0
WAN IOA util . . . . . :		.0
Integrated PC Server IOA util . . . . :	.0	.0
Interactive:		
CPU util . . . . . :	.0	.0
Int rsp time (seconds) . . . . . :	.0	.0
Transactions per hour . . . . . :	110	110
Non-interactive thrupt . . . . . :	58119	58135
Performance estimates -- Press help to see disclaimer.		
F3=Exit F6=Print F9=Work with spooled files F12=Cancel		

Figure 9-20 Verifying the BEST/1 model

7. Now you return to the Work with BEST/1 Model display (Figure 9-21). Choose option 7 to start exploring what would happen if the workload grew.

Work with BEST/1 Model	
Performance data . . . . :	QMPGDATA (TAKEONE)
Model/Text . . . . . :	YOURMODEL A descriptive text here
Select one of the following:	
1. Work with workloads	
2. Specify objectives and active jobs	
5. Analyze current model	
6. Analyze current model and give recommendations	
<b>7. Specify workload growth and analyze model</b>	
10. Configuration menu	
11. Work with results	
	More...
Selection or command	
==> <b>7</b>	
F3=Exit F4=Prompt F9=Retrieve F12=Cancel F15=Save current model	
F17=Analyze using ANZBESTMDL F22=Calibrate model F24=More keys	

Figure 9-21 Starting to grow the workload

8. When specifying the workload growth, you can grow the workload with a specified percentage for one growth period or multiple growth periods. You may specify separate growth percentages for separate growth periods as is shown in Figure 9-22. You may also allow BEST/1 to change the hardware configuration while growing the workload.

```

Specify Growth of Workload Activity

Type information, press Enter to analyze model.
Determine new configuration . . . . . Y      Y=Yes, N=No, K=Keep
                                         new configuration
Periods to analyze . . . . . 10    1 - 10

Period 1 . . . . . Period 1      Name
Period 2 . . . . . Period 2      Name
Period 3 . . . . . Period 3      Name
Period 4 . . . . . Period 4      Name
Period 5 . . . . . Period 5      Name

-----Percent Change in Workload Activity-----
Workload   Period 1  Period 2  Period 3  Period 4  Period 5
*ALL       .0       10.0     10.0     10.0     10.0

F3=Exit    F11=Specify growth by workload  F12=Cancel
F13=Display periods 6 to 10  F17=Analyze using ANZBESTMDL

```

The results are shown in the screen in Figure 9-23. Pay attention to Period 1 (our current workload) and especially the number of Non-Interactive transactions, which is 58135 in our case. Remember Figure 9-7 on page 190 where we chose the intervals to be used within the model and that during those intervals we created 800 orders? What do these two have in common? Answer: The number 58135 equals 800 orders generated.

Display Analysis Summary									
Period	CPU Model	Stor (MB)	-----CPU-----		DB Util	-Disk Nbr	IOPs- Util	-Disk Nbr	Arms-- Util
Period 1	270 23F5	4096	2/2	92.4	92.4	0	.0	12	3.3
Period 2	<b>820 23B8</b>	4096	4/4	73.4	73.4	0	.0	12	3.6
Period 3	820 23B8	4096	4/4	80.8	80.8	0	.0	12	4.0
Period 4	820 23B8	4096	4/4	88.8	88.8	0	.0	12	4.4
Period 5	820 23B8	4096	4/4	97.7	97.7	0	.0	12	4.8
Period 6	820 24B8	4096	4/4	90.4	90.4	0	.0	12	5.3
Period 7	820 24B8	4096	4/4	99.5	99.5	0	.0	12	5.8
Period 8	<b>830 23D8</b>	4096	8/8	54.7	54.7	0	.0	12	6.4
Period 9	830 23D8	4096	8/8	60.2	60.2	0	.0	12	7.0
Period10	830 23D8	4096	8/8	66.2	66.2	0	.0	12	7.7

Period	--Non-Inter Rsp Time--			-----Non-Inter-----		Release
	Local	LAN	WAN	CPU Util	Trans/Hr	Level
Period 1	.8	2.0	.0	92.4	<b>58135</b>	V5R1M0
Period 2	.2	2.7	.0	73.4	63949	V5R1M0
Period 3	.3	2.8	.0	80.7	70344	V5R1M0
Period 4	.4	2.8	.0	88.8	77378	V5R1M0
Period 5	1.8	2.8	.0	97.7	85116	V5R1M0
Period 6	.4	2.3	.0	90.4	93627	V5R1M0
Period 7	6.6	2.3	.0	99.5	102990	V5R1M0
Period 8	.2	2.3	.0	54.7	113289	V5R1M0
Period 9	.2	2.3	.0	60.2	124618	V5R1M0
Period10	.2	2.3	.0	66.2	137080	V5R1M0

F3=Exit    F10=Re-analyze    F11=Alternative view    F12=Cancel  
F15=Configuration menu    F17=Analyze multiple points    F24=More keys

Bottom

Figure 9-23 The results screen

Table 9-2 shows you the growing number of estimated transactions.

Table 9-2 Deciphering the real order lines from the BEST/1 transactions

BEST/1 orders generated	Real order lines generated in 30 min.	Real order lines generated per hour
63949 / 72.66875	880	1760
70344 / 72.66875	968	1936
77378 / 72.66875	1064	2128
85116 / 72.66875	1171	2342
93627 / 72.66875	1288	2576
102990 / 72.66875	1417	2834
113289 / 72.66875	1558	3116
124618 / 72.66875	1714	3428

BEST/1 orders generated	Real order lines generated in 30 min.	Real order lines generated per hour
137080 / 72.66875	1886	3772

## Modeling the workload for a different CPU model

Sometimes there comes a situation where you are not planning for growth, but trying to find the hardware that would support your workload. This scenario is described in the following steps:

1. Start modeling with either the STRBEST command or by navigating through the menus. In both cases, you end up with the Work with BEST/1 menu as shown in Figure 9-21 on page 198. After you press F21 to verify that you are using the Advanced user level, choose option **10** for the Configuration menu (Figure 9-24).
2. Choose option **1** to predict the effect of changing the CPU and the amount of Main Storage on the workload. You may use F4 to prompt for a list of values available.

Configuration			
CPU Model . . . . .	: 23F5	Comm IOPs . . . . .	: 0
Processors . . . . .	: 2/2	LAN lines . . . . .	: 2
Main stor (MB) . . . . .	: 4096	WAN lines . . . . .	: 0
Main stor pools . . . . .	: 6		
Disk IOPs . . . . .	: 0	Multifunction IOPs . . . . .	: 2
Disk ctls . . . . .	: 0	Disk IOAs . . . . .	: 2
Disk arms . . . . .	: 12	Comm IOAs . . . . .	: 2
ASPs . . . . .	: 1	IPCS IOAs . . . . .	: 0
Select one of the following:			
1. Change CPU and other resource values			
2. Work with disk resources			
3. Edit ASPs			
4. Edit main storage pools			
5. Work with communications resources			
Selection or command			
===>			
F3=Exit F4=Prompt F9=Retrieve F12=Cancel F13=Check configuration			
F17=Correct configuration F24=More keys			

Figure 9-24 The BEST/1 Configuration display

3. You can simply enter the values as shown in Figure 9-25.

```

Change CPU and Other Resource Values

Model/Text:  YOURMODEL  A descriptive text here

Type choices, press Enter.

System unit . . . . . 9406      9402, 9404, 9406
CPU Model . . . . . 24B8      F4 for list
System storage (MB) . . . . . 8192      F4 for list
Release level . . . . . V5R1M0      F4 for list
Active processors . . . . . 4
Unavailable PCI slots . . . . . 0

F3=Exit  F4=Prompt  F9=Specify other logical partitions  F12=Cancel

```

Figure 9-25 Specifying the CPU model and Main Storage size for the model

4. Changing the amount of Main Storage within the model leads you to the Edit Main Storage Pools display (Figure 9-26) where you must portion the additional memory between the memory pools. The easiest way to do this is to press the F17 key. This divides the memory evenly between the storage pools in the model. If you decide to specify the new pool sizes manually, you must make sure that the total amount of memory in the pools matches the amount of memory available before you can continue with using this model. We recommend that you use the re-scaling function, which is done by pressing the F17 key.

```

Edit Main Storage Pools

Model/Text:  YOURMODEL  A descriptive text here

Main storage size . . . . . :      8192  MB
                        8388608  KB

Type changes, press Enter.  To delete a pool, set size to *NOSTG.

Pool ID      Pool Name      Activity      Size
Level                               (KB)
1             0             479188
2             240           1005912
3             150           614400
4             5             41940
5             9             209664
6             300           1843200

Bottom

Activity level: 0=Unrestrained

F3=Exit  F6=Create  F12=Cancel  F17=Re-scale pool sizes

Sum of main storage pool sizes (4194304 KB) must equal system storage size

```

Figure 9-26 Re-scaling the memory pool sizes



5. After the new memory pool sizes are adjusted, you return to the main configuration display shown in Figure 9-27. On this display, you can both check that the configuration is valid and correct the configuration automatically, if needed.

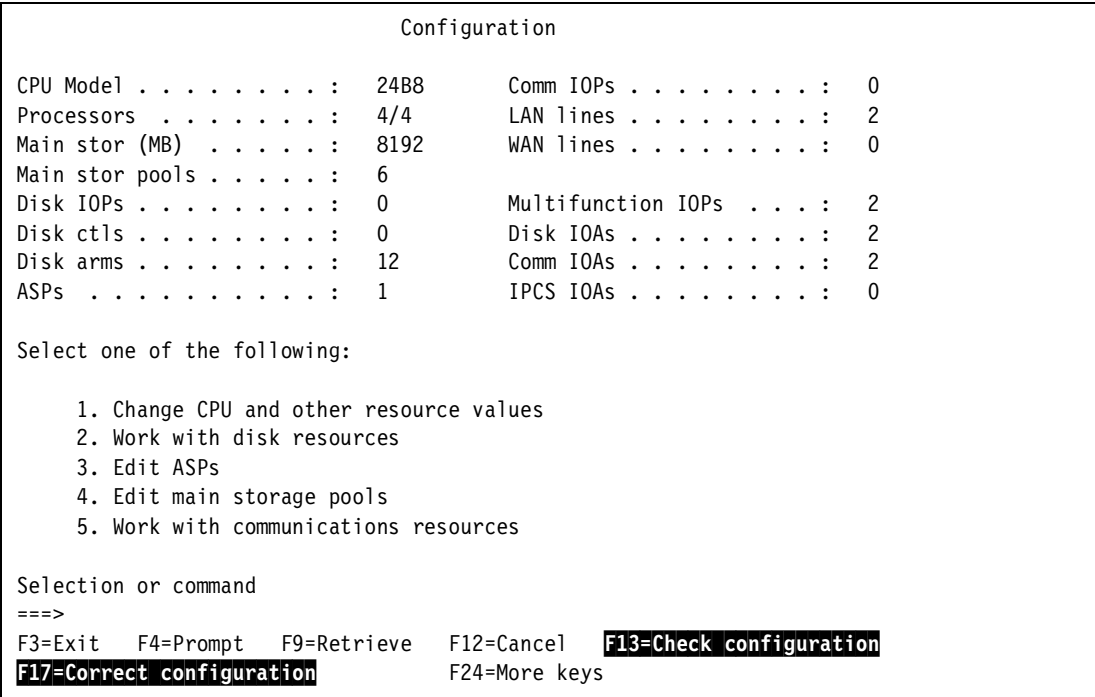


Figure 9-27 Checking and Correcting the configuration

Using the F17 key to correct the configuration provides you with a display similar to the one shown in Figure 9-28.

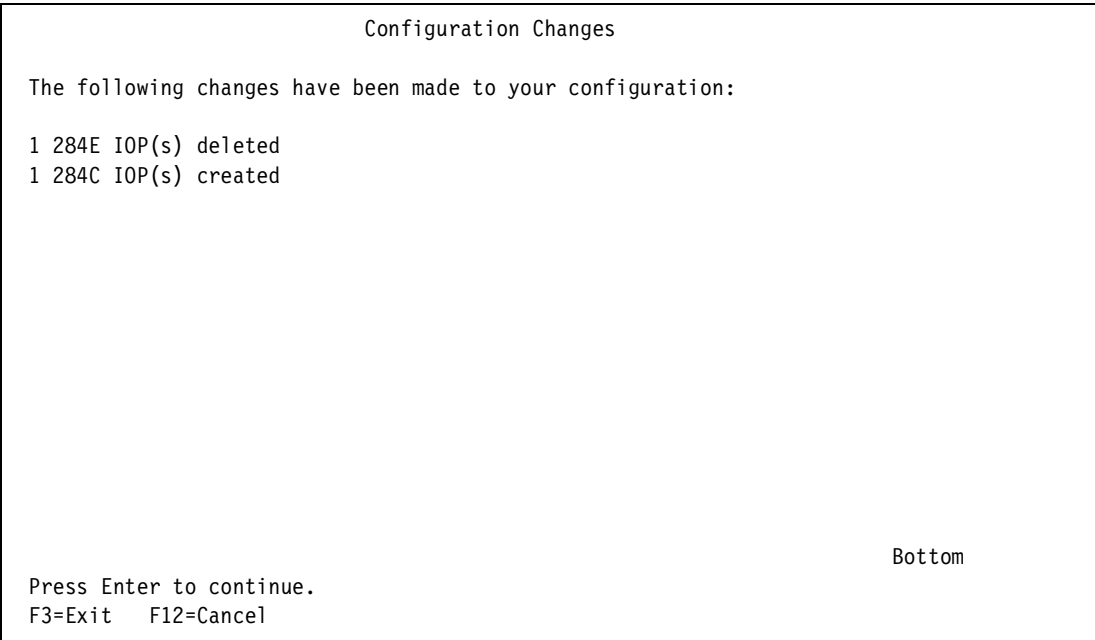


Figure 9-28 The configuration changes made by BEST/1

Entering option 5 to analyze current model on the Work with BEST/1 Model display provides you with the analysis summary that is shown in Figure 9-29. Press the F11 key (Measured and predicted comparison).

Display Analysis Summary			
CPU Model / release level	. . . . .	24B8	V5R1M0
Main Storage	. . . . .	8192	MB
		Quantity	Predicted Util
CPU	. . . . .	4/4	56.1
Database	. . . . .		56.1
Disk IOPs	. . . . .	0	.0
Disk ctls	. . . . .	0	.0
Disk arms	. . . . .	12	3.3
			More...
		Interactive	Non-interactive
CPU utilization %	. . . . .	.0	56.1
Transactions per Hour	. . . . .	110	58135
Local response time (seconds)	. . . . .	.0	.2
LAN response time (seconds)	. . . . .	.5	2.3
WAN response time (seconds)	. . . . .	.0	.0
Performance estimates -- Press help to see disclaimer.			
F3=Exit	F10=Re-analyze	F11=Measured and predicted comparison	F12=Cancel
F15=Configuration menu	F17=Analyze multiple points	F24=More keys	

Figure 9-29 Displaying the analysis summary with new CPU

6. On the Measured and Predicted Comparison screen (Figure 9-30), you can compare the measured values with the predicted values.

Measured and Predicted Comparison		
	Measured	Predicted
Total CPU util . . . . .	92.4	56.1
Database util . . . . .	92.4	56.1
Disk IOP util . . . . .	.0	.0
Disk arm util . . . . .	3.2	3.3
Disk IOs per second . . . . .	569.1	563.7
Multifunction IOP util . . . . .	2.3	2.4
Disk IOA util . . . . .	8.6	8.6
LAN IOA util . . . . .		.0
WAN IOA util . . . . .		.0
Integrated PC Server IOA util . . . . .	.0	.0
Interactive:		
CPU util . . . . .	.0	.0
Int rsp time (seconds) . . . . .	.0	.0
Transactions per hour . . . . .	110	110
Non-interactive thrupt . . . . .	58119	58135
Performance estimates -- Press help to see disclaimer.		
F3=Exit F6=Print F9=Work with spooled files F12=Cancel		

Figure 9-30 Comparing the measured data with the predicted model

### 9.3.2 IBM Workload Estimator

The IBM Workload Estimator tool provides both IBM and Business Partner sales/marketing people with a comprehensive iSeries sizing tool for new and existing customers that are interested in deploying new emerging workloads standalone or in combination with their current workloads. The recommendations provided by this tool are for the most recent hardware, and they project the model, CPU% utilization, memory, disk arms, and capacity.

It is easy to use, less than a dozen questions per application, with using default values for applications and system. You may specify workload types, workload details, and assess workload complexities before configuring the system. By making simple changes to the information already entered, you can evaluate the sensitivity to variations in workload of the proposed hardware.

A printed report with defaults, input parameters, and results is available.

The estimator is not a capacity planning tool or a configurator, since capacity planning tools offer detail modeling from actual system performance data. Capacity planning tools also project the response time for specific configurations.

Workload Estimator is basically a quick method for creating a system from scratch and it does not provide support for LPAR, journaling, response times, or locked resources.

To launch the Workload Estimator, point your browser to:

<http://as400service.ibm.com/estimator>

Using the Workload Estimator is straightforward. It requires a minimum amount of information, and it may be completed in a few steps as shown in the following sections.

► **Workload type selection**

The workload selection screen for the estimator is shown in Figure 9-31. You may specify multiple workloads. But remember that, during this process, you must describe each and every workload separately.

IBM Workload Estimator for iSeries - Netscape

File Edit View Go Communicator Help

Back Forward Reload Home Search Netscape Print Security Shop Stop

Bookmarks Location: http://www-912.ibm.com/servlet/EstimatorServlet

IBM WebMail Radio People Yellow Pages Download Calendar Channels

IBM Workload Estimator for iSeries  
an IBM eServer

Version: 2001.4  
16-Oct-01  
www-912

► File  
► Edit  
► Navigation  
Contact IBM  
► Tutorials  
► Help

## Workload Selection

Type of Workload	Name of Workload
Workload Type	Workload #1
Workload Type	Workload #2
Workload Type	Workload #3

Use the pull downs to select your workloads. Then press **Next**

Allow Another Workload

Options: OS/400 Version = V5R1, RAID Support = None, DBCS Support = No  
Developed by the Rochester iSeries System Performance Team

Figure 9-31 IBM Workload Estimator for iSeries

► **Workload detail entry**

Figure 9-32 shows how the specified workload volume and detail distribution may be entered. In this example, we only describe the WebSphere workload details, but you must describe all the workloads you specified on the previous page.

The screenshot shows a Netscape browser window titled "IBM Workload Estimator for iSeries - Netscape". The address bar shows the URL "http://www-912.ibm.com/servlet/EstimatorServlet". The browser's toolbar includes buttons for Back, Forward, Reload, Home, Search, Netscape, Print, Security, Shop, and Stop. Below the toolbar is a bookmarks bar with links to IBM, WebMail, Radio, People, Yellow Pages, Download, Calendar, and Channels. The main content area displays the "IBM Workload Estimator for iSeries" logo and the title "WebSphere #1 Workload Definition". On the left, there is a navigation menu with links for File, Edit, Navigation, Contact IBM, Tutorial, and Help. The main form contains five numbered questions with input fields or dropdown menus for answers. At the bottom of the form are "Back" and "Next" buttons. The status bar at the bottom of the browser window shows the current page and other navigation icons.

IBM Workload Estimator for iSeries  
an IBM® server

Version: 2001.4  
16-Oct-01  
www-912

WebSphere #1  
Workload Definition

1. [WebSphere Version](#)? ☐ v3.5 ☒ v4.0

2. How many [total visits per hour](#) do you anticipate for the server system during the **busiest** hour of the day?

3. In a typical visit, how many of the following operations will occur:

a. [Static web pages](#) served:

b. [Java Server Pages \(JSPs\)](#) served:

c. [Java Servlets](#) executed:

d. [EJB Session Beans](#) accessed:

e. [EJB Entity Beans](#) accessed:

4. [DBCS support](#) for this workload:

5. [RAID support](#) for this workload:

Back Next

Figure 9-32 Specifying the WebSphere workload

► **Workload complexity**

Figure 9-33 shows how the complexity of the WebSphere workload can be specified.


IBM Workload Estimator for iSeries - Netscape

File Edit View Go Communicator Help

Back Forward Reload Home Search Netscape Print Security Shop Stop

Bookmarks Location: <http://www-912.ibm.com/servlet/EstimatorServlet>

IBM WebMail Radio People Yellow Pages Download Calendar Channels

 **IBM Workload Estimator for iSeries**  
an IBM e server

Version: 2001.4  
16-Oct-01  
www-912

◀ ▶

File  
Edit  
Navigation  
Contact IBM  
Tutorials  
Help

# WebSphere #1

ExpertWorkload Definition

1. Rate the [complexity of the Java Servlets](#) used during a typical visit:
2. What is the [database utilization of the servlets](#)?
3. Rate the [complexity of the EJB Session Beans](#) used during a typical visit:
4. What is the [database utilization of the EJB Session Beans](#)?
5. Do your EJBs use [Pass-By-Reference](#)?

◀ Back Next ▶

Figure 9-33 Specifying the workload complexity

## ► Proposed configuration

Figure 9-34 shows the proposed configuration that is needed to support the previously described workload. Remember, the more accurately you describe the application functions and the workload, the more accurately the iSeries configuration is sized.

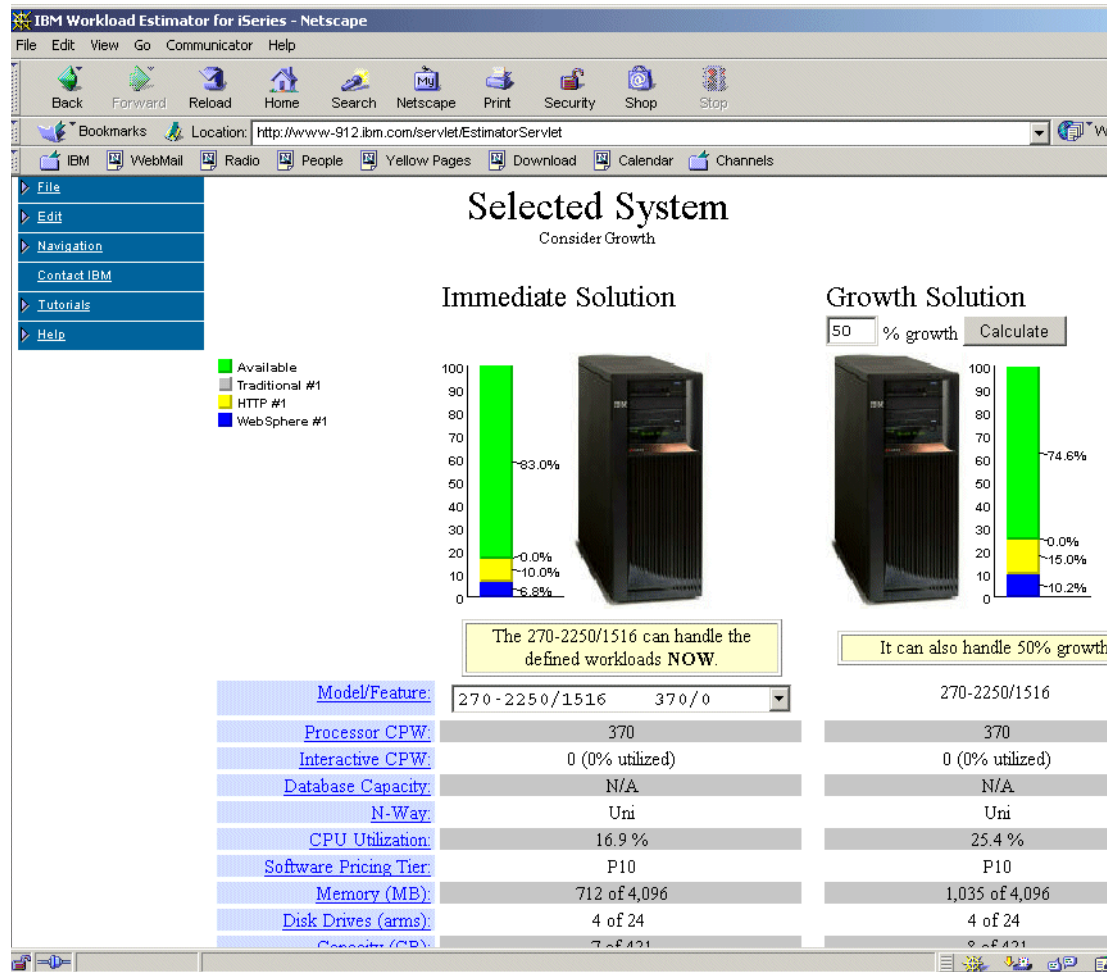


Figure 9-34 Estimated configuration

## Sensitivity to variations

By making simple changes to the details already specified, for example, workload complexity or growth, the estimated configuration can be recalculated to determine the effects of these changes. The more accurately you can describe the workload, the more accurately sized your system will be.

**Tip:** The data collected from your system with PM/400 is excellent input for Workload Estimator.

### 9.3.3 PM/400

PM/400 is a set of automated tools that collect performance data on your iSeries server and provide you with reports that are easy to understand. It is a tool that helps you determine the long-term workload growth and predict when a system upgrade may be needed. You also have the possibility of using the data collected with PM/400 as an input to the Workload Estimator.

For a full description of PM/400 services and features, see:

<http://www-1.ibm.com/servers/eserver/series/pm400/>

The only thing you have to know about PM/400 is to how to start it. Once it is started, it collects the proprietary system performance data on your iSeries server and downloads the data once a week to the factory machine for processing. The overhead generated by having the PM/400 running on your system is minimal.

**Tip:** The data collected with PM/400 is an excellent input for Workload Estimator.

## 9.4 Disk arm sizing

Insufficient number of disk arms can decrease the system performance. BEST/1 is the most reliable sizing tool for disk arms. In case you can't use BEST/1, use the disk arm sizing guidelines described in the document *iSeries Disk Arm Requirements Based on Processor Model Performance*, which is available on the Web at:

[http://www.as400.ibm.com/developer/performance/iSeries\\_Disk\\_Arm\\_Requirement.pdf](http://www.as400.ibm.com/developer/performance/iSeries_Disk_Arm_Requirement.pdf)

## 9.5 Stress testing the WebSphere application

While functional testing establishes the general correctness of an application, little indication of interactions between various application components can be determined without some form of volume or stress testing.

There are a few tools, commercial and open-source, that assist in this aspect of testing. It is not our intention to recommend or describe specific tools that may be generally available. Typically, such tools provide for the capture of certain detail from the client component of the application, the saving of this information, and the replaying in a manner which simulates multiple users. Some tools allow for either the random or sequenced change in values associated with a part of the captured data.

**Note:** The effort required to create a representative workload is significant, and the task should not be approached lightly.

### 9.5.1 General process for load and stress testing

The purpose of load testing is to generate a workload that acceptably represents the likely "real life" workload. It rarely does! However, in many cases, even a workload that appears significantly different, can provide the required degree of confidence that the more obvious interactions and bottlenecks have been investigated.

On the other hand, stress testing attempts to break the application by whatever means is practical and may involve degenerate transaction loads. In this context, a specific example of a degenerate transaction load is the same transaction generated by multiple virtual users.



In general, both load and stress testing may be used to locate bottlenecks *other than CPU*. In terms of WebSphere testing, the types of interactions that could be detected include:

- ▶ Inadequate number of application servers, such that queue sizes increase exponentially
- ▶ Synchronized code methods, such that most or all transactions lock (or bottleneck) on the code section
- ▶ Database locks
- ▶ Aspects of scalability

## 9.5.2 Load generation tools

As stated, there exist both commercial and open source tools, which generate a suitable load for testing. Alternatively there is also an option to roll-your-own, should the technical challenge appeal.

The general characteristics of such tools include the ability to:

- ▶ Record the details of a transaction manually executed
- ▶ Usually save the response sent back for each element of the transaction, in order to allow comparisons
- ▶ Generate a load by specifying the number of virtual users and the recorded scripts to be used
- ▶ Allow for the effects of think-time in the generated load
- ▶ Accumulate statistics during execution

We do not intend to describe the use of any particular tool.

## 9.6 Sizing an application under development

While it is relatively straightforward to determine apparently realistic numbers to put into a tool, such as IBM Workload Estimator, there is always a lingering doubt as to the way the application will perform in service, both from a response and a capacity perspective.

### 9.6.1 Validating the capacity estimates

Earlier we discussed the need for an ongoing measurement process during the full development lifecycle. We now extend that a little to outline a method where the first capacity estimates may be validated as development proceeds.

Since you have the capabilities of tools such as the WebSphere resource analyzer (see 3.6.1, “WebSphere Application Server Resource Analyzer” on page 38) and PTDV (see 3.7.3, “Performance Trace Data Visualizer for iSeries (PTDV)” on page 42), both of which provide measurement of CPU time used, then we may extend this to estimates of the production sizing.

The method that we suggest may be summarized as shown here:

1. Document the targeted business workload. It will not be correct, but can be refined later if required. This includes the business transactions by type and by relative frequency in the workload. Essential is also the business transaction rate/hour required.

2. Take the business transactions that comprise 80% of the workload, and document the mapping to server transactions. If there is an infrequent transaction, but one that can use significant resource, then map this as well.
3. Measure the CPU resource of the server transactions. Certainly this is not everything, but it is frequently less expensive to upgrade main storage or disk storage than the CPU. Extending the model to cover database IO is not particularly difficult.
4. From the mappings above, the total CPU seconds can be rolled up from server transactions to business transactions. Based on the rate/hour required, then the CPU seconds/second will indicate how close to the target you may be.
5. Now you are in a position to provide specific feedback to the transaction developers who have written code with characteristics exceeding the norm.

While simple to define, the above method requires some effort to implement. Once the initial setup is complete, then regular updating of transaction characteristics will allow easy assessment of the current development status against the sizing goals.

# 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 on ordering these publications, see “How to get IBM Redbooks” on page 215.

- ▶ *WebSphere V3 Performance Tuning Guide*, SG24-5657
- ▶ *WebSphere V3.5 Handbook*, SG24-6161
- ▶ *AS/400e Diagnostic Tools for System Administrators: An A to Z Reference for Problem Determination*, SG24-8253
- ▶ *AS/400 HTTP Server Performance and Capacity Planning*, SG24-5645
- ▶ *AS/400 Communication Performance Investigation - V3R6/V3R7*, SG24-4895
- ▶ *Advanced Functions and Administration on DB2 Universal Database for iSeries*, SG24-4249
- ▶ *IBM WebSphere Performance Pack: Load Balancing with IBM SecureWay Network Dispatcher*, SG24-5858

## Other resources

The publication, *OS/400 Work Management*, SC41-3306, is also relevant as a further information source.

## Referenced Web sites

These Web sites are also relevant as further information sources:

- ▶ **WebSphere**
  - An excellent list of publicly available white papers, covering a range of WebSphere related issues:  
<http://www.ibm.com/software/webservers/appserv/whitepapers.html>
  - WebSphere Resource Analyzer:  
<http://www.ibm.com/software/webservers/appserv/doc/v35/ae/infocenter/was/atsepm02.html>
  - Resource analyzer download site:  
[http://www.ibm.com/software/webservers/appserv/download\\_ra.html](http://www.ibm.com/software/webservers/appserv/download_ra.html)
  - iSeries WebSphere home page:  
<http://www.ibm.com/servers/eserver/series/software/websphere/wsappserver/>
  - WebSphere performance report:  
[http://www.ibm.com/software/webservers/appserv/was\\_performance.pdf](http://www.ibm.com/software/webservers/appserv/was_performance.pdf)

- WebSphere best practices:  
[http://www.ibm.com/software/webservers/appserv/ws\\_bestpractices.pdf](http://www.ibm.com/software/webservers/appserv/ws_bestpractices.pdf)
- “Methodology for Production Performance Tuning” white paper:  
[http://www.ibm.com/software/webservers/appserv/3steps\\_perf\\_tuning.pdf](http://www.ibm.com/software/webservers/appserv/3steps_perf_tuning.pdf)
- **IBM alphaWorks**
  - Provides access to new technologies available from IBM. This is also the download site for PTDV: <http://www.alphaworks.ibm.com>
  - Performance Trace Data Visualizer: <http://www.alphaworks.ibm.com/tech/ptdv>
- **iSeries Java**
  - The latest updates on Java for iSeries:  
<http://www.ibm.com/servers/eserver/iserries/java/newindex.htm>
  - The latest updates on IBM Toolbox for Java:  
<http://www.ibm.com/servers/eserver/iserries/toolbox/>
  - AS/400 Developer Kit for Java JDBC: <http://www.as400.ibm.com/developer/jdbc/>
  - *The iSeries Developer Kit for Java*:  
<http://publib.boulder.ibm.com/pubs/html/as400/infocenter.html>
- **iSeries performance and sizing**
  - Performance Explorer and PEXGUI (iDoctor):  
<http://www.ibm.com/servers/eserver/iserries/sftsol/pex.htm>
  - IBM Workload Estimator for iSeries:  
<http://www.as400service.ibm.com/servlet/EstimatorServlet>
  - iSeries Performance Management. Links to Management Central, Performance Tools/400, PM/400, iDoctor, PTDV, Workload Estimator:  
<http://www.ibm.com/servers/eserver/iserries/perfmgmt/>
  - iSeries Performance Capabilities Reference:  
<http://www-1.ibm.com/servers/eserver/iserries/perfmgmt/resource.htm>
- **Other iSeries related**
  - Provides access to the on-line Workload Estimator:  
<http://as400service.ibm.com/estimator>
  - iSeries White papers: <http://www.ibm.com/servers/eserver/iserries/whpapr/>
  - Using DB Monitor to identify and tune SQL queries:  
<http://www.as400.ibm.com/developer/bi/documents/dbmonitor.pdf>
- **IBM DeveloperWorks**
  - IBM DeveloperWorks: <http://www.ibm.com/developerworks/>

## How to get IBM Redbooks

Search for additional Redbooks or redpieces, view, download, or order hardcopy from the Redbooks Web Site

[ibm.com/redbooks](http://ibm.com/redbooks)

Also download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

Redpieces are Redbooks in progress; not all Redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

### IBM Redbooks collections

Redbooks are also available on CD-ROMs. Click the CD-ROMs button on the Redbooks Web Site for information about all the CD-ROMs offered, updates and formats.



# Abbreviations and acronyms

<b>API</b>	Application programming interface	<b>RMI</b>	Remote Method Invocation
<b>ASP</b>	Active Server Pages	<b>RPC</b>	Remote Procedure Call
<b>BCI</b>	AS/400 batch immediate (job)	<b>SLIC</b>	System Licensed Internal Code
<b>BCH</b>	AS/400 batch (job)	<b>SQL</b>	Structured Query Language
<b>CLI</b>	call level interface	<b>SSL</b>	Secure Socket Layer
<b>DBMS</b>	database management system	<b>TCP/IP</b>	Transmission Control Protocol/Internet Protocol
<b>DDM</b>	Distributed Data Management	<b>TIMI</b>	technology independent machine interface
<b>DE</b>	direct execution	<b>TSE</b>	time slice end
<b>DPC</b>	Distributed Program Call (also RPC)	<b>UDB</b>	Universal Database
<b>EJB</b>	Enterprise JavaBeans	<b>UOW</b>	unit of work
<b>EPM</b>	Extended Program Model (AS/400)	<b>URL</b>	uniform resource locator
<b>EPM</b>	enable performance monitoring (WebSphere Application Server)	<b>URI</b>	uniform resource identifier
<b>GC</b>	garbage collection or garbage collector	<b>WCBT</b>	Work Control Block Table
<b>GUI</b>	graphical user interface	<b>WPO</b>	Whole Program Optimization
<b>HTML</b>	Hypertext Markup Language	<b>WTE</b>	WebSphere Test Environment
<b>HTTP</b>	Hypertext Transfer Protocol	<b>WWW</b>	World Wide Web
<b>IBM</b>	International Business Machines Corporation	<b>XML</b>	eXtensible Markup Language
<b>ITSO</b>	International Technical Support Organization		
<b>JAR</b>	Java archive		
<b>JDBC</b>	Java Database Connectivity		
<b>JDK</b>	Java Developer's Kit		
<b>JFC</b>	Java foundation classes		
<b>JIT</b>	Just-in-time Java compiler		
<b>JNI</b>	Java Native Interface		
<b>JNDI</b>	Java Naming and Directory Interface		
<b>JPDC</b>	Java Performance Data Converter		
<b>JSDK</b>	Java Servlet Development Kit		
<b>JSP</b>	JavaServer Pages		
<b>JVM</b>	Java Virtual Machine		
<b>MI</b>	Machine Interface		
<b>ODP</b>	open data path		
<b>PEX</b>	Performance Explorer		
<b>PJ</b>	prestart job		
<b>PTDV</b>	Performance Trace Data Visualizer		
<b>RDBMS</b>	relational database management system		





# Index

## A

- activity level 70, 71
- Add PEX Definition 46
- additional number of active jobs 65
- additional number of total jobs 64
- ADDPDXDFN 46
- Advisor 37
  - conclusions 37
  - recommendations 37
- AKstress 127
- allocation system values 61
- alphaWorks 43
- AnyNet 81
- Apache 91
- application 2
- application-level tools 39
- architecture, three-tier versus two-tier 132
- ASP
  - capacity 26
  - usage 26
- assistance level 62
- auto reload 98
- AutoCommit 119
- automatic tuning 60
- AWT 114

## B

- base storage pool activity level 65
- base storage pool minimum size 66
- batch immediate job 9
- batch insert 119
- BCI job 9
- Beans.instantiate() 123
- BEST/1 187
- BEST/1 configuration 201
- BigDecimal 119
- blocking READ\_ONLY, WRITE\_ONLY 120
- browser 3
- business-to-business sites 171
- bytecode 9

## C

- caching 84
  - EJB Home interfaces 155
- Call By Reference 107
- Call By Value 107
- call level interface (CLI) 116
- capacity planning 181, 182, 186
- CCSID 118
- CCSID 13488 118
- Change System Value (CHGSYSVAL) command 61
- character fixed-length string 113
- CHGPJE 79, 80

- CHGRTGE 69
- CHGSBSD 68
- CHGSYSVAL (Change System Value) command 61
- CLI (call level interface) 116
- client performance 2
- closed queue versus open queue 93
- CLOSQLCSR 118
- coding considerations 112
- collecting performance data 19
  - for analysis 48
- Collection Services 25
- command framework 125
- communication lines 5, 38
- communications IOPs 5
- configuration
  - checking 203
  - correcting 203
- connection pool size 96
- connection pooling 174
- CPA4072 49
- CPU 4
- CPU utilization 37
- Create Java Program (CRTJVAPGM) command 10, 44
- Create Performance Data 22, 35
- CRTJVAPGM (Create Java Program) command 10, 44
- CRTPFRDTA command 35, 187
- CUM PTF package 58
- Cumulative Information 49
- Cumulative Procedure Information 50
- customer self-service sites 170
- customer table 132
- Customer Table Layout (CSTMR) 140

## D

- data conversion 118
- database 114
- database access using JDBC 157
- Database Monitor 51
- Database Monitor for iSeries 51
- database server 2
- database tuning 118
- DataSources 123
- DB Monitor 51
- DDM (Distributed Data Management) 14, 115, 120
- DE (direct execution) 9, 11, 44, 91
- DE versus JIT 91
- debug 105
- direct execution (DE) 9, 11, 44, 91
- dirty read 160
- disk 38
  - protection status 29
- disk arm 5
  - sizing 210
- disk IOPs 5

- disk space 5
- Display Active Prestart Jobs (DSPACTPJ) command 75
- Display Data Area (DSPDTAARA) command 58
- Display Java Program (DSPJVAPGM) 44
- Display Job Tables (DSPJOBTL) command 62
- Display Performance Data (DSPPFRTA) command 36
- Distributed Data Management (DDM) 14, 115
- Distributed Procedure Call (DPC) 14
- Distributed Program Call (DPC) 115, 121
- district table 132
- District Table Layout (Dstrct) 139
- DMPJVM (Dump Java Virtual Machine) command 40
- domain object 134
- DPC (Distributed Procedure Call) 14
- DPC (Distributed Program Call) 115, 121
- DSPACTPJ (Display Active Prestart Jobs) command 75
- DSPDTAARA (Display Data Area) command 58
- DSPJOBTL (Display Job Tables) command 62
- DSPJVAPGM (Display Java Program) command 44
- DSPPFRTA (Display Performance Data) command 36
- Dump Java Virtual Machine (DMPJVM) command 40
- dynamic caching 84
- dynamic SQL 116, 117

## E

- EBCDIC 118
- EJB coding considerations 124
- EJB container 94, 102
- EJB deployment 125
- EJB homes 125
- EJB isolation levels 159
- EJB workload management 179
- ENBPFRCOL 10
- End Database Monitor (ENDDBMON) parameters 52
- End PEX (ENDPEX) command 42
- entity beans 125, 132
  - application development scenario 132
- Entity Enterprise JavaBeans 132
- EPM (Extended Performance Monitoring) 107
- Event Summary 49
- exception handling 113
- Expert Cache 67, 73
- extended dynamic SQL 117
- extended dynamic support 116
- Extended Performance Monitoring (EPM) 107

## F

- failover 168
- fault isolation 169
- fault-tolerance 168
- fixed-length string 113

## G

- garbage collection 11, 112
  - initial size 97
- get...(columnIndex) 119
- get...(columnName) 119
- group PTF 58

## H

- hardware 2
- horizontal scaling 176, 178
- Host Servers 13
- HTTP server 2, 83
- HTTP server threads 95
- HttpServlet Init() method 122, 156
- HttpSessions 121, 122

## I

- IBM network dispatcher (eND) technology 174
- IBM Toolbox for Java 13, 115
- IBM Workload Estimator 205
- initial heap size 12
- initial number of active jobs 64
- initial total number of jobs 62
- instantiation 112
- integrated DB2 database 176
- IOPs 38
- iSeries Host Servers 13
- iSeries performance behavior 58
- iSeries performance measurement tools 19
- iSeries scaling considerations 175
- iSeries WebSphere scaling 177
- isolation errors 160
- isolation level 126, 159
- item table 132
- Item Table Layout (ITEM) 141

## J

- java command 9
- Java Database Connectivity (JDBC) 14, 115
- Java Development Kit (JDK) 8
- Java implementation components 8
- Java Native Interface (JNI) 113
- Java performance 5
- Java program 10
- Java runtime components 9
- Java thread priority 81
- Java Transformer 9
- Java Virtual Machine 2
- javac -O 113
- JDBC (Java Database Connectivity) 14, 115, 117, 118
- JDBC driver limitations 180
- JDK (Java Development Kit) 8
- JIT (just-in-time) compiler 9, 11
- JIT mode versus DE 91
- JLOG 123
- JNDI 122
- JNI (Java Native Interface) 113
- Job/Thread List 49
- JSP 121
- JSP Engine 104
- just-in-time compiler (JIT) 9, 11
- JV1 8
- JVM 2

## L

load balancing 168  
load generation tools 211  
load testing 210

## M

machine storage pool size 67  
main memory 5, 67  
main storage 5, 37, 67  
Management Central 25  
management collection 35  
manual tuning 60  
MAXACT 65  
maximum activity level of system 66  
maximum number of active HTTP server threads 88  
MAXRCDS 49  
memory 2  
memory leak 40  
memory pool 67  
    creating 67  
    determine size 70  
    private 67  
    shared 67  
    sizes 70  
method resolution 113  
mixed configuration 169  
model  
    changing CPU 201  
    creating 188  
    growth 195

## N

native calls 14  
native driver 116, 117  
network components 3  
network dispatcher 177  
network dispatcher (eND) technology 174  
network performance 3  
No Local Copies 107  
non-repeatable read 160

## O

object diagram 129  
Object Information 50  
object model 134  
ODBC environment 116  
OLTP (online transaction processing environment) 118  
online shopping sites 170  
online transaction processing environment (OLTP) 118  
open and close 120  
open queue versus closed queue 93  
operating system 3, 58  
Operations Navigator 21  
optimization level 10  
Order Line Table Layout (ORDLIN) 141  
order table 132  
Order Table Layout (ORDERS) 140  
OrderEntry application 128

OrderEntryClerk 130  
OrderEntryClerk EJB 132, 133, 134  
OrderPlacement 131  
OrderPlacement EJB 132, 133  
OS/400 system tuning 60  
outer transaction context 125

## P

page fault 70  
paging  
    CALC 73  
    FIXED 73  
performance activity profile 186  
Performance Explorer 41  
performance goals 16  
performance measurement tools 19  
performance methodology 15, 183  
performance objectives 16  
performance process 183  
performance strategy 17  
Performance Tools for iSeries 35  
Performance Trace Data Visualizer 42  
persistent connection 86  
persistent session 101, 177  
PEX 41  
phantom read 160  
ping initial timeout 99  
ping interval 107  
ping timeout 99  
plug-in to HTTP server 104  
PM/400 210  
ports 36  
Prepared statement 158  
prepared statement 120  
prepared statement cache 103  
prestart jobs 74  
Print PEX Report (PRTPEXRPT) command 42  
problem determination 18  
processor speed 2  
PTDV 41, 42  
    collecting data for 43  
    prepare for use 43  
    using 44  
PTF package 58  
publish/subscribe Web sites 170

## Q

QACTJOB 61, 64  
QADLACTJ 61, 65  
QADLSPLA 61  
QADLTOTJ 61, 64  
QAPEXDFN 41  
QAQQDBMN 52  
QBASACTLVL 65  
QBASPOOL 65, 66  
QCMN subsystem 74  
QEJBADV4 74  
QEJBAES4 74  
QEJBSBS 73, 144

- QHTTPSVR 74
- QINTER 74
- QJOBMSGQFL 61
- QJOBMSGQMX 61
- QJOBMSGQSZ 61
- QJOBMSGQTL 61
- QJOBSPLA 61
- QJVACMDSRV 74
- QMAXACTLVL 65, 66
- QMAXJOB 61
- QMAXSPLF 61
- QMCHPOOL 65, 67
- QPEXDATA 42
- QPFRADJ 61
- QRCLSPLSTG 61
- QRWTSRVR 14, 74
- QSERVER 74
- QSERVER subsystem 74
- QSQSRVR 14, 74, 75, 80
- QSQSRVR jobs 79
- QSYSWRK 74, 143
- QTOTJOB 61, 62
- queue
  - closed queue versus open queue 93
  - settings 94
  - tuning 94
- queuing network 93
- QZDASOINIT 14, 74, 75, 116
- QZRCRSRV 14, 74, 115

## R

- RAM (Random Access Memory) 67
- Random Access Memory (RAM) 67
- READ\_ONLY 120
- ReadAll() Method 121
- read-only method 125
- record level access 120
- record-level access 115
- Redbooks Web Site 215
  - Contact us xi
- reload interval 98
- Remote AWT 114
- remote databases 180
- remote method calls 125
- Resource Analyzer 38
- retrieving by index 158
- retrieving by name 158
- rollback 143
- RPG order entry application
  - application flow 134
  - customer transaction 133
  - database 129
  - tables 139
- RUNJVA 9
- RUNPTY setting 81

## S

- scalability 168, 171
  - objectives 168

- scaling the WebSphere environment 167
- Secure Sockets Layer (SSL) 87
- security 102, 169
- server components 3
- server performance 3
- servlet clustering 180
- servlet engine 108
- servlet queue size 95
- Servlet Redirector 93
- servlets 121, 136
- session bean 125, 132
  - CPWEJB application 132
  - stateful 125, 133
  - stateless 133
- Session Enterprise JavaBeans 132
- Session Manager 101
- SingleThreadModel 122
- sizing an application 211
- sockets 14
- SQL extended dynamic support 116
- SQL Package 116
- SQL statement cache 116
- SQL Visual Explain 53
- SQLJ 117
- SSL (Secure Sockets Layer) 87
- standard error 100
- standard output 100
- Start BEST/1 (STRBEST) command 188
- Start Database Monitor (STRDBMON) command 52
- Start Performance Tools (STRPFRT) command 19
- Start PEX (STRPEX) command 42
- statement cache 117
- static SQL 117
- stock table 132
- Stock Table Layout (Stock) 141
- Stop and Copy 12
- storage pools 67
- storage system values 65
- stored procedures 118
- STRBEST (Start BEST/1) command 188
- STRDBMON (Start Database Monitor) command 52
- stress testing 210
- string manipulation 113
- StringBuffer 113
- STRPFRT (Start Performance Tools) command 19
- system level tools 25
- system performance tools 37
- system services 114
- system tuning 145
- system values 61
- System.out.println() 122

## T

- thread creation 114
- thread priority 81
- thread state 65
- three-tier architecture 132
- time slice 187
- time slice setting 81
- trace 105

- tracing 105
- trading sites 170
- transaction isolation attribute 160
- transaction isolation level 126
- TRANSACTION\_READ\_COMMITTED 160
- TRANSACTION\_READ\_UNCOMMITTED 126, 160, 161
- TRANSACTION\_REPEATABLE\_READ 160
- TRANSACTION\_SERIALIZABLE 126, 160, 161
- transition data 72
- tuning process 17
- tuning the application 150
- tuning the HTTP server 84
- tuning the queues 94
- tuning WebSphere Application Server 145
- two-phase commit 180
- two-tier 132
- two-tier architecture 132
- type 2 driver 116
- type 4 driver 116

## U

- Unicode 118

## V

- validating capacity estimates 211
- variable scope 114
- vertical scaling 178
- Visual Explain 53

## W

- WCBT (Work Control Block Table) 61
- Web application server 2
- Web server 2
- WebSphere Application Server 2
  - Resource Analyzer 38
  - tuning 145
- WebSphere environment, scaling 167
- WebSphere performance tools 38
- WebSphere queuing network 93, 94
- WebSphere Resource Analyzer 38
- WebSphere scalability 176
- Whole Program Optimization (WPO) 10
- Work Control Block Table (WCBT) 61, 217
- Work with Active Jobs 30
- Work with Disk Status 27
- Work with System Activity 34
- Work with System Status 26
- Work with System Value (WRKSYSVAL) command 61
- workload
  - complexity 208
  - detail 207
  - specifying growth 199
  - type 206
- workload estimation tools 186
- Workload Estimator 205
- workload patterns 169
- WPO (Whole Program Optimization) 10

- WRITE\_ONLY 120
- WRKACTJOB 30
  - command 30
  - find 33
- WRKDSKSTS command 27
- WRKOBJLCK command 38
- WRKSHRPOOL command 68
- WRKSYSACT command 34
- WRKSYSSTS command 26, 72
- WRKSYSVAL (Work with System Value) command 61













**Redbooks**

# Java and WebSphere Performance

on IBM @server iSeries Servers

**Understand the Java and WebSphere environment on iSeries**

**Use the proper tools for performance analysis and planning**

**Learn performance tuning tips and techniques**

This IBM Redbook provides tips, techniques, and methodologies for working with Java and WebSphere Application Server performance-related issues with a specific focus on iSeries servers. The intended audience includes iSeries system performance experts and WebSphere Application Server and Java developers. This book attempts to provide you with a comprehensive and accessible resource that has gathered information usually found in a number of disparate sources and placed them in a single reference.

The performance measurements that are published in this redbook are done by using a Java application that runs on WebSphere Application Server Release 3.5.4 and 4.0 and OS/400 V5R1. This redbook includes these topics:

- ▶ An introduction to the Java and WebSphere Application Server execution environment on the iSeries
- ▶ OS/400 work management and tuning in Java/WebSphere Application Server environment
- ▶ Tuning WebSphere Application Server and Web server (HTTP server)
- ▶ Performance tips for application development and deployment
- ▶ Sizing and capacity planning
- ▶ Recommended tools to use for performance analysis and planning
- ▶ A case study that shows an example of performance management methodology and tuning

**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](http://ibm.com/redbooks)

SG24-6256-00

ISBN 0738423645