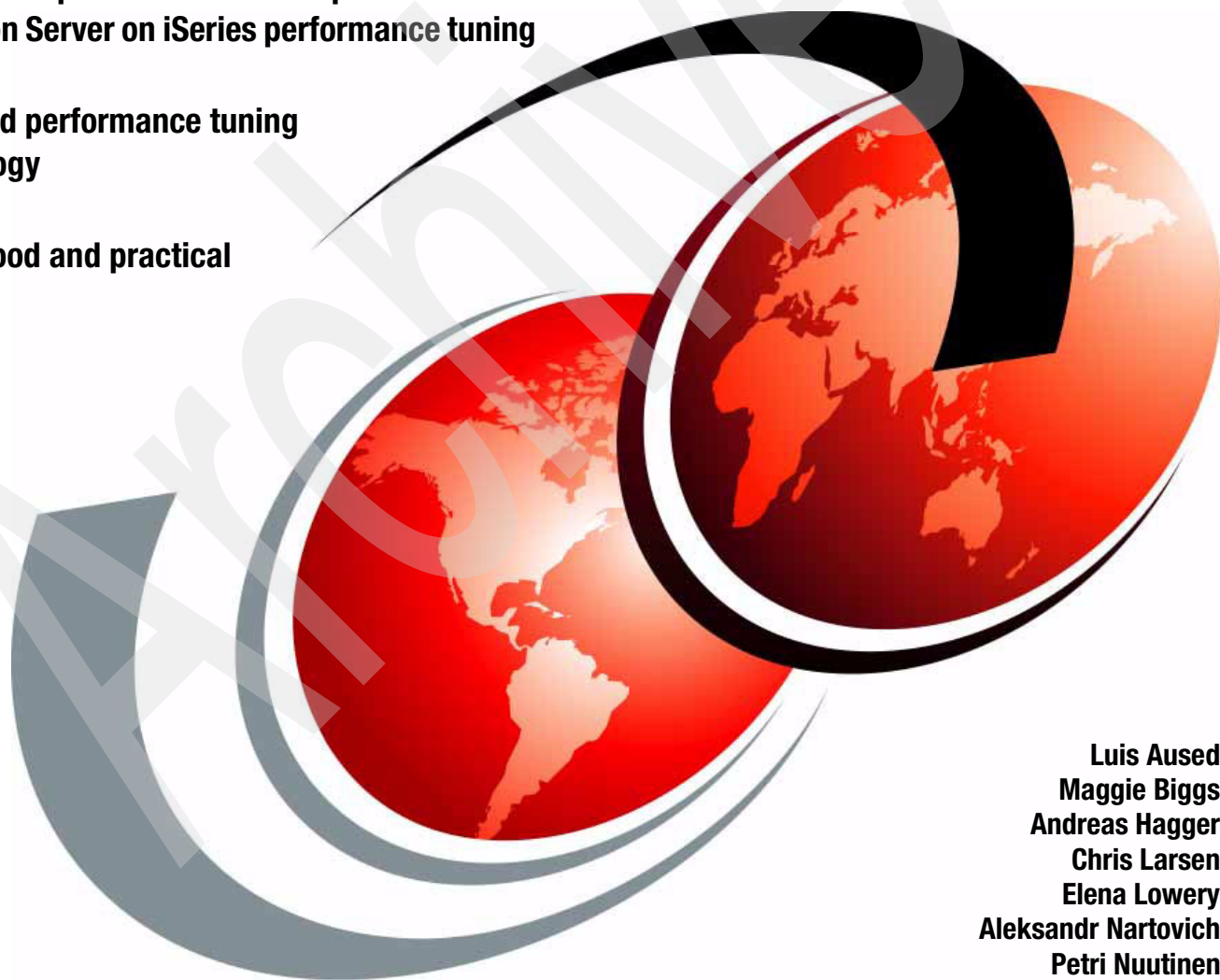


Maximum Performance with WebSphere Application Server V5.1 on iSeries

The most complete book on WebSphere
Application Server on iSeries performance tuning

End-to-end performance tuning
methodology

Several good and practical
examples



Luis Aused
Maggie Biggs
Andreas Hagger
Chris Larsen
Elena Lowery
Aleksandr Nartovich
Petri Nuutinen

Redbooks



International Technical Support Organization

**Maximum Performance with WebSphere
Application Server V5.1 on iSeries**

January 2005

Archived

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

Archived

First Edition (January 2005)

This edition applies to Version 5, Release 1 of WebSphere Application Server on iSeries (5733-W51).

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

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

Contents

| | |
|---|------|
| Notices | ix |
| Trademarks | x |
| Preface | xi |
| The team that wrote this redbook. | xi |
| Become a published author | xiii |
| Comments welcome. | xiii |
| Chapter 1. Performance concepts and factors | 1 |
| 1.1 Topology | 2 |
| 1.2 Client considerations | 4 |
| 1.3 Network design | 5 |
| 1.4 Maximizing server performance | 6 |
| 1.5 Performance and emerging architectures | 8 |
| 1.5.1 Service-oriented architectures | 8 |
| 1.5.2 Virtualization | 9 |
| Chapter 2. Java and WebSphere execution environment on iSeries servers | 11 |
| 2.1 Java execution environment | 12 |
| 2.1.1 Java virtual machine implementation | 12 |
| 2.1.2 Java Runtime Environment | 12 |
| 2.2 Java compilation environment | 12 |
| 2.2.1 Direct execution using Java transformation | 13 |
| 2.2.2 JIT compiler | 13 |
| 2.2.3 Compiling and running Java code | 14 |
| 2.3 Garbage collection | 15 |
| 2.3.1 Basics in collecting garbage | 15 |
| 2.3.2 Garbage collection pitfalls | 16 |
| 2.4 Profiling (PFRDTA) | 16 |
| 2.4.1 Profiling on the iSeries | 16 |
| 2.4.2 Collecting profiling data | 16 |
| 2.5 IBM Toolbox for Java | 17 |
| 2.6 WebSphere execution environment | 17 |
| 2.6.1 Administration | 18 |
| 2.6.2 Application development | 18 |
| 2.6.3 Security | 19 |
| 2.6.4 Web services | 20 |
| Chapter 3. Performance tuning methodology | 21 |
| 3.1 Setting performance objectives | 22 |
| 3.2 General tuning process and performance strategy | 23 |
| 3.3 Performance problem determination process | 24 |
| 3.4 Summary of key performance measurement tools | 25 |
| Chapter 4. Tools for determining performance problems | 27 |
| 4.1 System-level tools | 28 |
| 4.2 System-level tools for real time use | 28 |
| 4.2.1 WRKSYSSTS command | 28 |
| 4.2.2 WRKDSKSTS command | 29 |

| | | |
|-------------------|--|------------|
| 4.2.3 | WRKACTJOB command | 31 |
| 4.2.4 | WRKSYSACT command | 35 |
| 4.2.5 | Management Central | 36 |
| 4.2.6 | iDoctor | 40 |
| 4.2.7 | HEAPANA command | 81 |
| 4.2.8 | Database Monitor for iSeries | 85 |
| 4.2.9 | Identifying and tuning problem areas | 91 |
| 4.2.10 | SQL Visual Explain | 91 |
| 4.3 | System-level tools for reactive use | 93 |
| 4.3.1 | Collecting performance data | 93 |
| 4.3.2 | Creating performance data | 96 |
| 4.3.3 | Performance Tools for iSeries | 98 |
| 4.3.4 | Using the system performance tools | 99 |
| 4.3.5 | Performance Management for iSeries, PM/400, and PM eServer iSeries | 100 |
| 4.4 | Java and application-level tools | 101 |
| 4.4.1 | DMPJVM command | 101 |
| 4.4.2 | ANZJVM command | 102 |
| 4.4.3 | JAVAGCTOOLS (STRSST) | 106 |
| 4.5 | WebSphere performance tools | 116 |
| 4.5.1 | Profiling applications running on the WebSphere Application Server | 117 |
| 4.5.2 | Performance Monitoring Infrastructure | 128 |
| 4.5.3 | Tivoli Performance Viewer | 130 |
| 4.5.4 | Performance Advisors | 132 |
| 4.5.5 | Log Analyzer | 142 |
| 4.6 | HTTP performance tools | 145 |
| 4.6.1 | Server logging | 146 |
| 4.6.2 | HTTP server tracing | 149 |
| 4.6.3 | Collection Services | 152 |
| 4.6.4 | Communications trace | 156 |
| 4.7 | Stress testing tools | 157 |
| 4.7.1 | IBM Rational stress testing tools | 159 |
| 4.7.2 | OpenSTA | 162 |
| Chapter 5. | Tuning iSeries for a WebSphere or Java environment | 165 |
| 5.1 | iSeries performance behavior | 166 |
| 5.2 | Verifying the state of Licensed Program Products | 166 |
| 5.3 | OS/400 system tuning | 168 |
| 5.3.1 | Manual tuning versus automatic tuning | 168 |
| 5.3.2 | Semi-automatic system tuning | 168 |
| 5.3.3 | Verifying the settings of system values | 169 |
| 5.3.4 | Creating a separate memory pool for a subsystem | 174 |
| 5.3.5 | Setting the memory pool sizes and activity levels manually | 177 |
| 5.3.6 | Working with the prestart jobs | 183 |
| 5.4 | Java system environment | 188 |
| 5.4.1 | Java thread run priority | 189 |
| 5.4.2 | Java thread time slice setting | 189 |
| 5.5 | AnyNet | 190 |
| 5.6 | Example of an OS/400 tuning process | 190 |
| Chapter 6. | Tuning the IBM HTTP Server (powered by Apache) | 199 |
| 6.1 | Web server performance | 200 |
| 6.1.1 | The server | 200 |
| 6.2 | Caching | 200 |

| | |
|--|------------|
| 6.2.1 Dynamic and local (static) caching | 201 |
| 6.2.2 Proxy caching | 201 |
| 6.2.3 Tuning cache | 202 |
| 6.2.4 Fast Response Cache Accelerator | 203 |
| 6.2.5 Network File Cache | 203 |
| 6.3 Persistent connections | 204 |
| 6.4 Secure Sockets Layer and Transport Layer Security | 204 |
| 6.5 HTTP server tuning options | 204 |
| 6.5.1 Threads and asynchronous I/O | 204 |
| 6.5.2 Triggered Cache Manager | 207 |
| 6.5.3 mod_deflate | 208 |
| 6.5.4 Logging | 208 |
| 6.5.5 HostNameLookups | 210 |
| 6.5.6 KeepAliveTimeout | 210 |
| 6.5.7 TCP/IP buffer sizes | 210 |
| 6.5.8 Denial of service | 211 |
| 6.6 Static content location | 212 |
| 6.7 References | 215 |
| Chapter 7. Tuning the Java virtual machine | 217 |
| 7.1 Java virtual machine on iSeries | 218 |
| 7.1.1 High level view of JVM on the iSeries server | 218 |
| 7.1.2 Runtime modes of execution | 219 |
| 7.2 Garbage collection in iSeries JVM | 223 |
| 7.2.1 Understanding garbage collection | 224 |
| 7.2.2 Garbage collection performance tuning | 224 |
| 7.2.3 Problems with garbage collection | 230 |
| Chapter 8. Tuning WebSphere Application Server Base | 231 |
| 8.1 Adjusting WebSphere Application Server system queues | 232 |
| 8.1.1 Queue configuration tips | 233 |
| 8.1.2 Enterprise bean method invocation queuing | 237 |
| 8.2 WebSphere Application Server tuning parameters | 238 |
| 8.2.1 Web container | 239 |
| 8.2.2 Session management | 244 |
| 8.2.3 Dynamic cache service | 247 |
| 8.2.4 Data sources | 258 |
| 8.2.5 EJB container | 264 |
| 8.2.6 Object Request Broker | 266 |
| 8.2.7 Additional performance tips | 272 |
| 8.3 Java Messaging Service tuning parameters | 278 |
| 8.3.1 Listener service | 279 |
| 8.3.2 Listener port | 280 |
| 8.3.3 JMS resources | 281 |
| 8.4 Tuning your security configuration | 283 |
| Chapter 9. Java and WebSphere application design | 285 |
| 9.1 Java general design | 286 |
| 9.1.1 String manipulation | 286 |
| 9.1.2 Object instantiation | 288 |
| 9.1.3 Loops | 289 |
| 9.1.4 Exceptions | 290 |
| 9.1.5 Method resolution | 290 |
| 9.1.6 Logging | 291 |

| | |
|--|------------|
| 9.1.7 Variables | 292 |
| 9.1.8 Collections | 292 |
| 9.1.9 Java Native Interface | 293 |
| 9.1.10 Thread creation | 294 |
| 9.2 Accessing system services: Database operations | 294 |
| 9.2.1 IBM Toolbox for Java driver versus iSeries Developer Kit for Java driver | 295 |
| 9.2.2 JDBC through a native driver | 296 |
| 9.2.3 Which one to use? | 298 |
| 9.2.4 Coding considerations with JDBC | 298 |
| 9.2.5 Caching your data source | 307 |
| 9.3 Coding considerations with DDM (record-level access) | 308 |
| 9.3.1 Minimizing open and close | 308 |
| 9.3.2 Blocking on READ_ONLY and WRITE_ONLY | 308 |
| 9.3.3 Downloading a small file using the readAll() method | 308 |
| 9.4 Distributed Program Call | 309 |
| 9.5 Coding consideration for servlets and JSPs | 309 |
| 9.5.1 Do not store large objects in HttpSession | 309 |
| 9.5.2 Releasing HttpSession when finished | 309 |
| 9.5.3 Do not create HttpSession in JSPs by default | 310 |
| 9.5.4 Do not use SingleThreadModel | 310 |
| 9.5.5 Using the HttpServlet init() method | 310 |
| 9.5.6 Using the HttpServlet destroy() method | 311 |
| 9.5.7 Minimizing or eliminating the use of System.out.println() | 311 |
| 9.5.8 Don't use Beans.instantiate() to create new bean instances | 312 |
| 9.5.9 Using and reusing data sources for JDBC connections | 312 |
| 9.5.10 Releasing JDBC resources when done | 314 |
| 9.5.11 Page generation using strings | 314 |
| 9.5.12 Consideration with static content | 315 |
| 9.5.13 JSP include directive versus Action | 315 |
| 9.5.14 Minimizing the useBean scope | 316 |
| 9.6 Coding considerations for EJBs | 316 |
| 9.6.1 Accessing entity beans from session beans | 316 |
| 9.6.2 JNDI names | 317 |
| 9.6.3 Reusing EJB home interfaces | 319 |
| 9.6.4 Using info beans to reduce remote calls | 321 |
| 9.6.5 Releasing resources in EJBs | 323 |
| 9.6.6 Using local interfaces when possible | 323 |
| 9.6.7 Removing stateful session beans when finished | 324 |
| 9.6.8 Transactions | 324 |
| 9.6.9 Using CMPs instead of BMPs | 327 |
| 9.6.10 Message-driven beans | 327 |
| 9.6.11 Using home methods | 327 |
| 9.6.12 Using the setEntityContext() method | 328 |
| 9.6.13 Don't use entity beans as simple data wrappers | 328 |
| 9.6.14 Minimizing the use of stateful session beans | 328 |
| 9.6.15 Using Data Class Access Beans | 328 |
| 9.7 Profiling your application | 330 |
| 9.7.1 Method speed | 330 |
| 9.7.2 Finding memory leaks | 337 |
| 9.8 Using a response time watcher | 342 |
| Chapter 10. Scaling and capacity planning for WebSphere applications | 345 |
| 10.1 Scalability objectives | 346 |

| | | |
|--------|---|------------|
| 10.2 | Scaling a solution | 347 |
| 10.2.1 | Workload patterns | 347 |
| 10.2.2 | Scalability in practice | 349 |
| 10.2.3 | Six steps to scaling your solution | 350 |
| 10.3 | iSeries-specific scaling considerations | 353 |
| 10.3.1 | iSeries advantages and configuration customization options | 354 |
| 10.3.2 | WebSphere for iSeries scalability overview and key concepts | 354 |
| 10.3.3 | JDBC driver choices on the iSeries | 358 |
| 10.4 | Planning future capacity to maximize performance | 358 |
| 10.5 | Generic performance process and methodology | 359 |
| 10.5.1 | Process steps | 360 |
| 10.6 | Capacity planning and workload estimation tools | 362 |
| 10.6.1 | IBM Workload Estimator | 363 |
| 10.6.2 | PM eServer iSeries | 367 |
| 10.7 | Disk arm sizing | 368 |
| 10.8 | Stress testing WebSphere applications | 368 |
| 10.9 | Sizing an application under development | 368 |
| | Appendix A. Additional material | 371 |
| | Locating the Web material | 371 |
| | Using the Web material | 371 |
| | System requirements for downloading the Web material | 371 |
| | How to use the Web material | 372 |
| | Related publications | 373 |
| | IBM Redbooks | 373 |
| | Other publications | 373 |
| | Online resources | 373 |
| | How to get IBM Redbooks | 374 |
| | Help from IBM | 374 |
| | Index | 375 |

Archived

Notices

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

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

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

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

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

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

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



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

COPYRIGHT LICENSE:

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

Trademarks

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

| | | |
|--|-------------------------|---|
|  eServer® | ClearCase® | OS/400® |
| alphaWorks® | ClearQuest® | PurifyPlus™ |
| ibm.com® | COBOL/400® | Rational® |
| iSeries™ | Domino® | Redbooks (logo)  ™ |
| i5/OS™ | DB2 Universal Database™ | Redbooks™ |
| pSeries® | DB2® | RequisitePro® |
| zSeries® | Footprint® | Tivoli® |
| AnyNet® | IBM® | WebSphere® |
| AS/400e™ | Lotus® | |
| AS/400® | MQSeries® | |

The following terms are trademarks of other companies:

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

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

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

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

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

Preface

There may be several different reasons why you want to read this book. Perhaps you are currently experiencing a performance issue with an application. Or, you may be designing and developing a new application that you want to deploy on the IBM® WebSphere® Application Server running on the IBM @server iSeries™ server.

This book helps you to gain a solid understanding of the factors that can affect the performance of the Java™ and WebSphere applications running on the iSeries server. As a deployment platform, the iSeries architecture combined with Java and the WebSphere Application Server provides a robust execution environment, which is ideal for the mission-critical business applications and Web sites.

This IBM Redbook explains how to tune the iSeries server for IBM WebSphere Application Server. The book is divided into several logical tuning tasks, such as tuning the Java virtual machine. You can study each of these logical blocks and apply them independently from other tasks. However, to achieve optimum performance on the system, we highly recommend that you use the systematic approach that is shown in this book.

This book is written for the system administrators, application developers, and I/T consultants who are already familiar with WebSphere Application Server and work management topics for the iSeries server. If you're not proficient in these topics, you should read other IBM Redbooks™ and manuals, some of which are referenced in this IBM Redbook.

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 (ITSO).

Luis Aused Lopez is an IT Architect for IBM Global Services in Spain, working under IBM Business Consulting Services (BCS) in travel and transportation. He has worked for IBM for over eight years. During this time, he has developed applications for the iSeries server, including a Data Warehouse application for an insurance company. Over the last four years, he developed several On Demand Business ticketing applications, for Spanish railways, using WebSphere on the IBM @server zSeries® platform, which interface with DB2® and MQSeries®. His area of expertise includes application development for On Demand Business, the iSeries server, WebSphere, DB2 and eTicketing. Luis holds a degree in physics from Complutense University, Madrid, Spain.

Maggie Biggs is a senior engineer with Countrywide Financial Corporation. Her current duties include the performance management of a large WebSphere Application Server environment running across multiple iSeries systems. She has more than 20 years of multiplatform technology experience in a variety of roles, including architect and software developer.

Andreas Hagger is a Software Engineer and member of the IBM EMEA WebSphere Support Organization, based in Switzerland. He has over 17 years of experience in working with S/36 and iSeries servers. His areas of expertise include Java, WebSphere and database. You can contact him by sending e-mail to ah@ch.ibm.com.

Chris Larsen is President of KnowledgeGain Inc., an IBM Business Partner and Solution Provider based in Rochester, Minnesota (MN). In his 16 years experience as a Sr. Enterprise

Architect, Chris' primary responsibility is to provide consulting services centered around Java 2 Enterprise Edition (J2EE) technologies and WebSphere. His areas of expertise on the iSeries focus on WebSphere application design, performance tuning, capacity planning, and legacy migration strategies. Chris writes extensively and is the author of several IBM Redbooks and technical publications. He holds bachelor degrees in electronics engineering and computer science. You can contact Chris by sending e-mail to Chris@KnowledgeGain.com.

Elena Lowery is a Technical Consultant in the IBM @server Solutions Enablement organization at IBM Rochester. She helps IBM Business Partners implement various WebSphere technologies on the iSeries platform. Her areas of expertise include WebSphere Application Server, WebSphere Portal Server, Web services, and Java development.

Aleksandr Nartovich is a Senior I/T Specialist in the IBM ITSO, Rochester Center. He joined the ITSO in January 2001 after working as a developer in the IBM WebSphere Business Components (WSBC) organization. During the first part of his career, Aleksandr was a developer in AS/400® communications. Later, he shifted his focus to business components development on WebSphere. Aleksandr holds two degrees: in computer science from the University of Missouri-Kansas City and in electrical engineering from Minsk Radio Engineering Institute. You can reach Aleksandr by sending e-mail to alekn@us.ibm.com.

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 responsible for the technical support of the AS/400 MRI translation process. His technical specialties are work management and performance tuning. He has participated in several performance-related residencies and redbook projects since 1991. Petri currently divides his working time between pre and post sales and billable services.

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

Jim Cook
Thomas Gray
ITSO, Rochester Center

Michael R. Burke
Steve Fullerton
Jay Kurtz
Scott Moore
Paul M Swenson
IBM Rochester, Minnesota

Christian Griere
IBM France

Become a published author

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

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

ibm.com/redbooks

- Send your comments in an Internet note to:

redbook@us.ibm.com

- Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. JLU Building 107-2
3605 Highway 52N
Rochester, Minnesota 55901-7829

Archived

Performance concepts and factors

To maximize the performance of Java and WebSphere Application Server on the iSeries, you must have a clear understanding of the components that comprise your application. Ideally, performance should receive major consideration from application design time onward throughout the development life cycle. You can greatly reduce production-level performance issues by implementing coding standards that follow best practices to maximize runtime performance. Chapter 9, “Java and WebSphere application design” on page 285, explains how to design applications with performance in mind.

During runtime, performance can also be impacted by factors other than your application's code. Other core infrastructure components that can affect performance include:

- ▶ Topology
- ▶ Clients
- ▶ Network
- ▶ Servers

Maintaining expected application runtime performance requires that you understand, plan, and closely manage the performance settings of all of your core infrastructure components.

If your application runtime includes requirements, such as high availability, you also need to size and tune it for expected failover and load balancing requirements in a clustered setting. Moreover, if you are moving from a legacy application-oriented architecture (AOA) to a service-oriented architecture (SOA) that may include a virtualized deployment, you must size and retune your application's performance attributes for the new architecture.

1.1 Topology

The topology you choose for your application runtime can greatly impact performance. In short, it can be the difference between success and failure.

By topology, we mean the servers and network connections that you will use within your infrastructure to execute your application. Java and WebSphere applications consist of several server-side technologies:

- ▶ **Java virtual machine (JVM):** The JVM provides the runtime environment for Java programs. On iSeries servers, the JVM is part of the operating system.
- ▶ **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 closed.

- ▶ **Web application server:** This server provides the essential On Demand Business functions of handling transactions and extending back-end business data and applications to the Web. The IBM Web application server is WebSphere Application Server, which is covered in this redbook. WebSphere Application Server supports servlets, JavaServer Pages (JSPs), Enterprise JavaBeans (EJBs), and Web services. It provides a Java-based servlet engine that's built on top of the iSeries native Java virtual machine (JVM). It implements the Java servlet application programming interface (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.
- ▶ **Database server:** This is usually a relational database management system (RDBMS). On the iSeries server, the database server is DB2 Universal Database™ (UDB) for iSeries, which is also a part of the operating system.
- ▶ **Application:** This is the set of modules that executes your business logic, such as order entry, general ledger, and so on.

Figure 1-1 shows the server-side components that comprise a WebSphere Application Server-based application runtime.

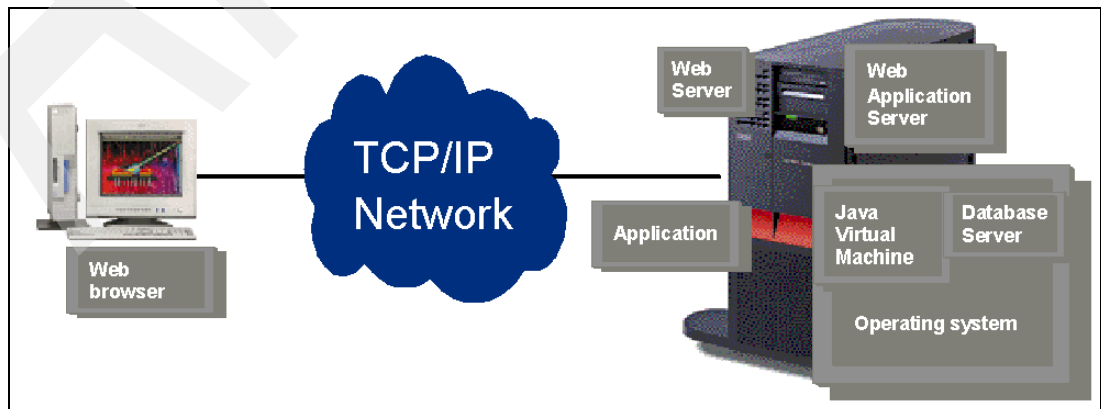


Figure 1-1 WebSphere application runtime

Topologically speaking, your application can be deployed in a number of different types of configurations. In the simplest form, you may have a single machine which executes the Web server (HTTP), application server (WebSphere Application Server), and the database server.

Figure 1-2 illustrates a single machine configuration that executes the Web, application, and database servers.

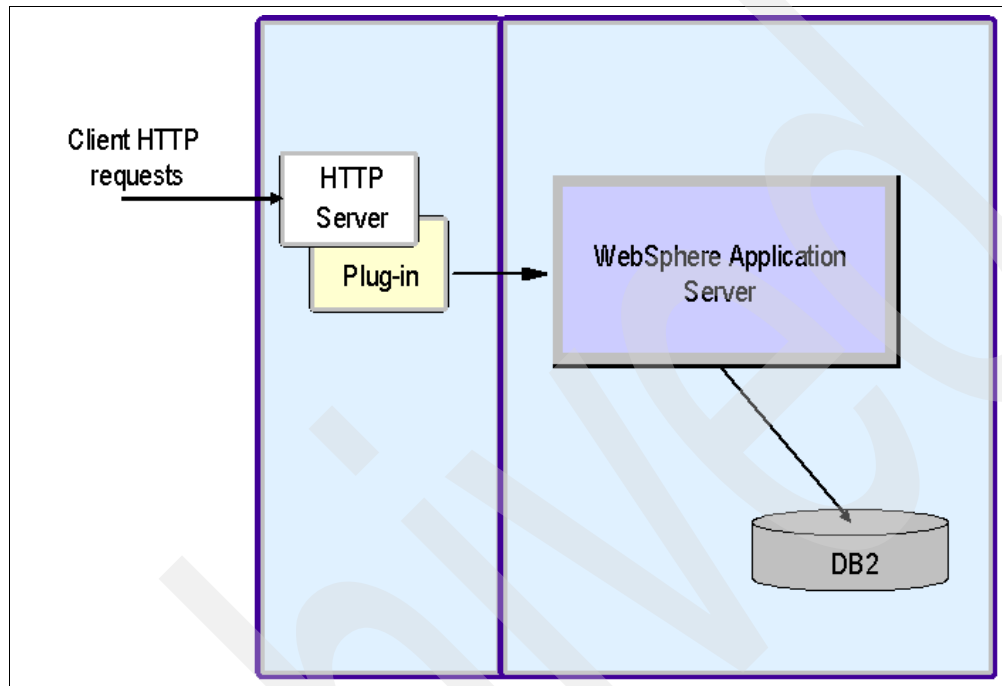


Figure 1-2 Single machine configuration

Depending on your performance objectives, a single machine configuration may meet your requirements perfectly. However, there may be instances where a multimachine topology is a much better fit. For example, if you expect a high volume of client requests, you may use a multimachine configuration to maximize performance under heavy load.

Multimachine topologies are also a good match for high availability environments where application requirements demand constant uptime. In this scenario, the Web, application, and database components can be implemented using multiple machines, so that expected or unexpected server downtime does not impact the application runtime.

Figure 1-3 shows an example of a multi-machine configuration where the Web and application server layers include redundant capabilities.

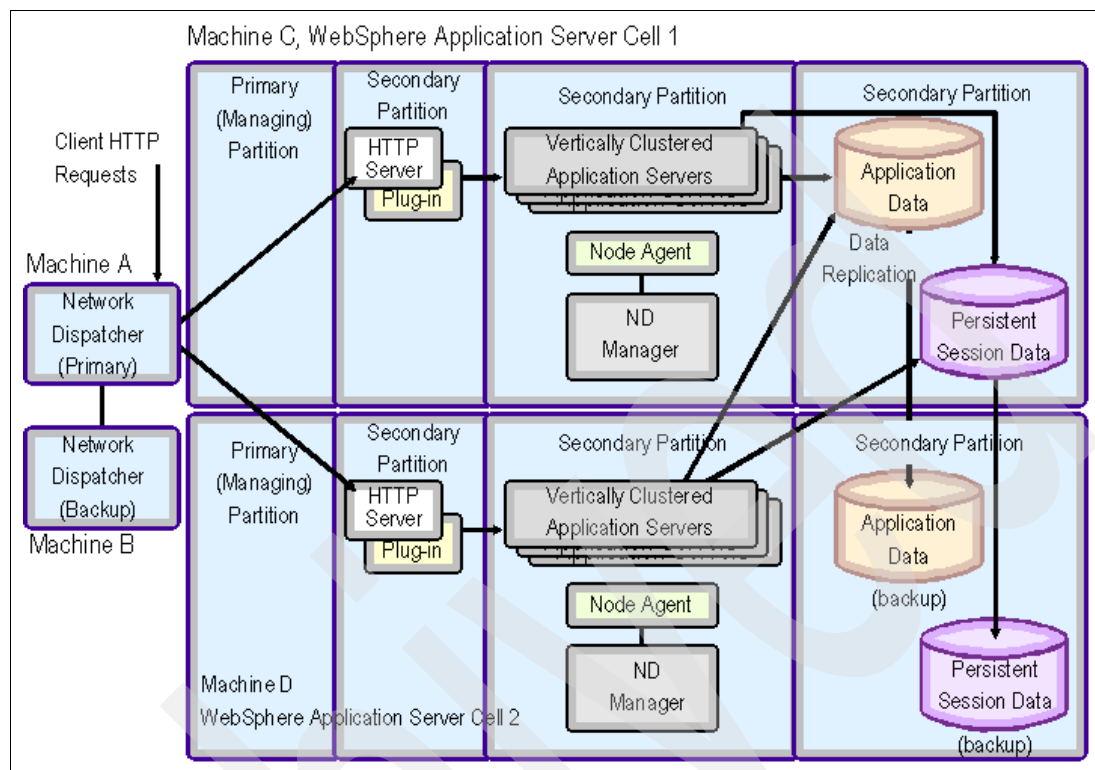


Figure 1-3 Redundant WebSphere topology

This redbook does not provide a comprehensive discussion about topologies. When planning your application runtime environment, refer to the following sources for more information:

- ▶ *IBM WebSphere V5.1 Performance, Scalability, and High Availability*, SG24-6198
- ▶ *WebSphere Application Server V5 for iSeries: Installation, Configuration, and Administration*, SG24-6588
- ▶ *IBM WebSphere Developer Technical Journal: Planning for Availability in the Enterprise*
http://www-106.ibm.com/developerworks/websphere/techjournal/0312_polozoff/polozoff.html
- ▶ WebSphere Application Server for iSeries Information Center
<http://publib.boulder.ibm.com/was400/51/english/index.htm?info/rzaiz/51/was.htm>

For more information about scaling your WebSphere Application Server configuration to maximize performance, see Chapter 10, “Scaling and capacity planning for WebSphere applications” on page 345.

1.2 Client considerations

Most of the processing in the applications deployed within a WebSphere Application Server environment is executed on the server side. Therefore, the role of the client (workstation, mobile computer) is typically limited to support the use of a Web browser, such as Mozilla Firefox, Netscape Navigator, or Microsoft® Internet Explorer.

From this point of view, the client's impact on the overall application performance is less significant than the performance of a network or a server. Still, the client contributes to the system's performance (good or bad) with the following client resources:

► **Hardware:**

- *Processor speed:* Slower clients, such as those with 333 or 400 MHz central processing unit (CPU) speeds, may experience performance degradation compared to faster clients, such as those with 1500 MHz or greater CPU speeds. We recommend that you upgrade your client to use the fastest processor speeds.
- *Memory:* Memory is an important factor since many Web-related tasks can use large amounts of memory. 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 may not be fast enough, or for dial-up users, modem speed may not be fast enough. To maximize utilization of these resources, upgrade your hardware. In some cases, you can merely 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 difficulties, consider using a different operating system if you think this is the reason for the delay in the response time.

- **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 as possible. This helps you to reduce client requests to the server.

1.3 Network design

Usually, the network has much more of an impact on performance than the client. This is due to a wide variety of factors, including throughput, network traffic, and the speed of communication links. You can examine these elements in greater detail if you look at such network components as:

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

The network is a dynamic environment. Often you cannot control all of 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 about network performance, refer to the following sources:

- *iSeries Performance Capabilities Reference Version 5, Release 2*, SC41-0607
- *AS/400 HTTP Server Performance and Capacity Planning*, SG24-5645

1.4 Maximizing server performance

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

The server performance components are integrated in three major areas:

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

Figure 1-4 outlines the general composition of the server. However, you may find that additional components are needed in your own environment. You need to fully understand and be aware of these components to maximize server performance.

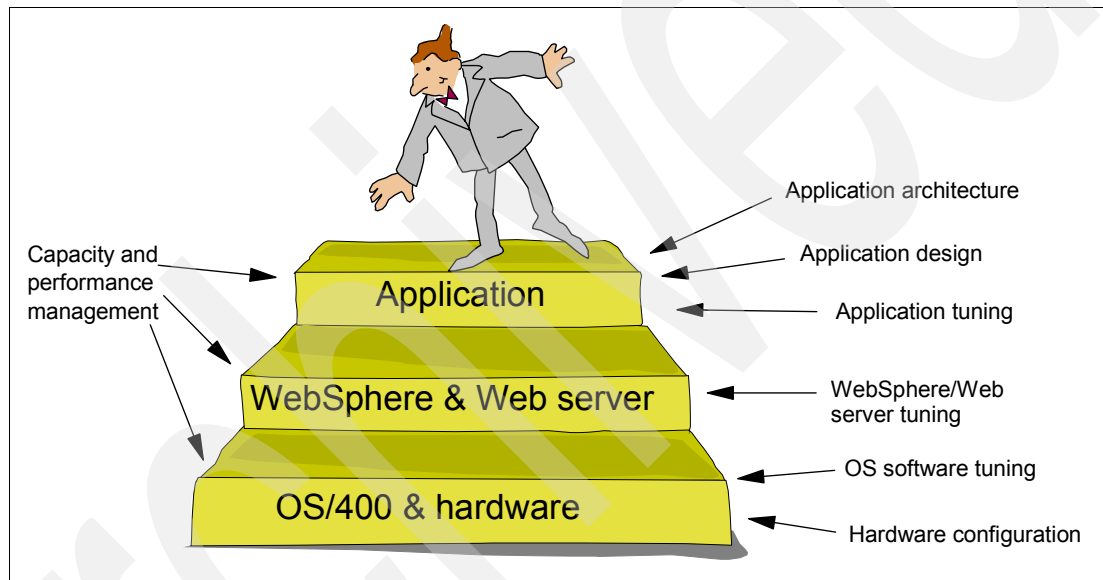


Figure 1-4 Performance as a balancing act

Performance is a balancing act. 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 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:* 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.

Consider all these elements 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 applications are typically CPU intensive, use the servers with fast CPUs. We recommend that you use iSeries Models 270, 5xx, and 8xx. Look for the servers with higher MHz, not only a higher central processing workload (CPW). If your iSeries server runs only Java and WebSphere Application Server workloads, 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 that you have a server with a level 2 (L2) cache. The bigger the L2 cache is, the better your results are.
- ▶ **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 (I/O) 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. When you have less than 50% of free disk space, disk service times start to increase. When you have less than 10% of free disk space, the system starts sending you warning messages. For good performance, try to keep your disk arm utilization under 50% and disk IOP utilization under 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. Much has happened to the Java language since it was first released by Sun in May 1995. It was originally developed to support the creation of applets for interactive Web browser content (and actually designed for embedding in devices, such as toasters). Since then, Java has undergone a transformation into a full featured programming language for business applications. The promise of platform independence and an easy language syntax have proven a great attraction to many 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 solutions for On Demand Business. With this came an increased focus on component-based development. Also application servers became an important part of the infrastructure, providing standardized persistence mechanisms (Java Database Connectivity (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. In some cases, it can prove inefficient or simply not work. A good deployment methodology, system tuning, and application design should always be the first steps 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 (best practices) can take some of the guesswork out of the performance of your Java application.

If you are developing a new application, stress test and check the performance and scalability of your early prototype during unit testing. Keep stress testing on a regular basis as your application develops. In particular, include several rounds of stress testing during the user-acceptance test phase of development. This is one of the key methods for successful deployment with minimum surprises.

1.5 Performance and emerging architectures

The majority of this redbook focuses on maximizing the performance of applications deployed to the WebSphere Application Server in client/server architectures. Client/server configurations are commonplace within most companies and nearly ubiquitous for Web-based applications. However, you may also want to examine emerging architectures, such as a service-oriented approach or virtualization as additional options with which to maximize performance.

1.5.1 Service-oriented architectures

The WebSphere Application Server includes support for building and deploying Web services. A *Web service* is a self-contained, modular application that you describe, publish, locate, and invoke over a network.

Web services reflect a new, service-oriented approach to development. In an SOA, applications are built by discovering and implementing network services or by invoking available applications to accomplish some task.

SOAs are not dependent on specific programming languages or operating systems. Web services deliver the capability for components created in different programming languages to work together as though they were created using the same language. Moreover, Web services rely on existing transport technologies, such as HTTP or Java Messaging Service (JMS), and standard data encoding techniques, such as Extensible Markup Language (XML), for invoking the implementation.

There are three roles in an SOA:

- ▶ **Service provider:** Creates components, and then publishes them to a repository. Within the WebSphere Application Server platform, these components include:
 - Enterprise beans
 - Java beans
 - DB2 UDB stored procedures
 - Server-side scripts that implement the Bean Scripting Framework (BSF)Service providers can also unpublish components (remove them from the repository) when they are no longer needed.
- ▶ **Service broker:** Categorizes Web services as they are published and provides a search mechanism for the services as requests are received. Web brokers are similar to an Internet search engine, except that they locate components instead of Web pages.
- ▶ **Service requester:** Look up, or locate and invoke components as services. They act as the client for published Web services.

Figure 1-5 shows process interaction in an SOA.

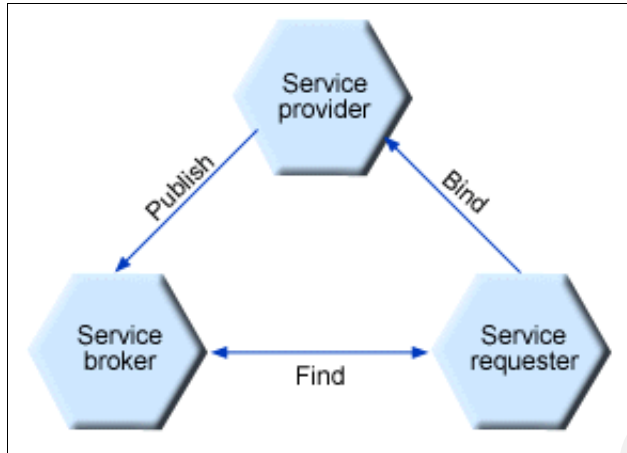


Figure 1-5 Interaction in a service-oriented architecture

Like AOA, SOAs require a multi-layer approach to maximize performance. However, there are additional performance considerations, such as parsing, that are specific only to Web services deployments.

1.5.2 Virtualization

Another new architectural choice is the construct of virtualization. The goal is to create what appears to be a simple, yet large and powerful self-managing virtual computer from a collection of connected heterogeneous systems.

In a virtualized environment, you can unite pools of servers, storage systems, and networks into one large system to deliver non-trivial qualities of service. The virtualized resources may be in the same room, or distributed across the globe; running on multiple hardware platforms; using different operating systems; and potentially owned by different organizations. To an end user or application, the virtualized resources look like one big computing system.

Figure 1-6 illustrates a simple virtualized environment that can be used internally within a company. More complex virtualized configurations can be used internally or between business partners.

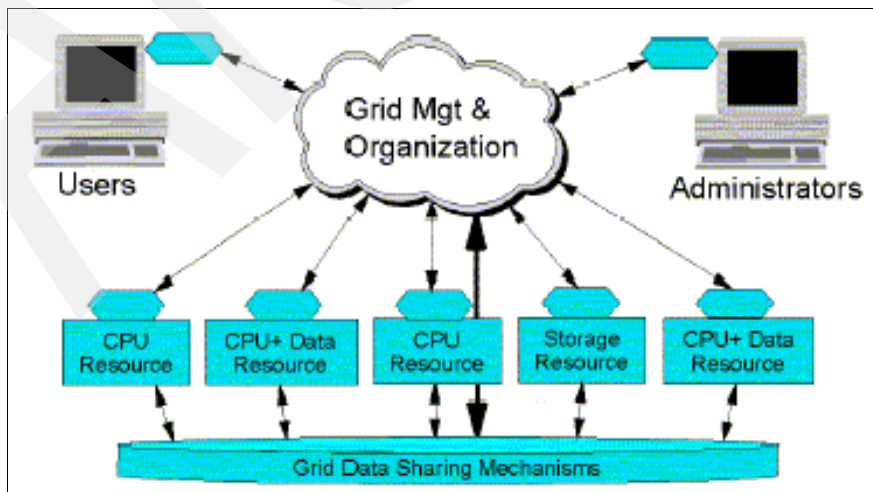


Figure 1-6 Simple virtualized environment

Such technologies as clusters, network attached storage, scientific devices, or networks are not virtualized themselves. Yet each of these technologies may be an important component within a virtualized configuration. When you construct your virtualized environment, you define which resources to make available.

Here are some examples of how a virtualized environment may be used:

- ▶ **Computational tasks:** Machines with spare resources stand by to crunch data or provide coverage for other intensive workloads.
- ▶ **Resource scavenging:** This is commonly used to salvage spare CPU cycles from idle servers and desktop machines for use in resource-intensive tasks.
- ▶ **Data-related activity:** This provides a unified interface for all of the disparate data repositories in an organization, and through which data can be queried, managed, and secured.

As with AOA and SOA, a virtualized runtime requires a layered approach to maximize performance. You must be concerned with the hardware, system, and application (or service) level performance, as well as the combined performance of all of the shared resources.

In particular, virtualized runtimes require careful attention to tracking the distribution of work among the shared resources. Failure to manage work distribution properly can lead to bottlenecks in the virtualized environment. Conversely, when managed properly, a virtualized environment can provide a powerful mechanism with which to scale your applications and Web services as business needs grow and change.

Regardless of the type of architecture you use to deploy your applications and Web services, this redbook takes you through some of the secrets to unlocking superior Java and WebSphere performance from your iSeries server.

Java and WebSphere execution environment on iSeries servers

At this point in the Java revolution, everyone has either embraced Java or reluctantly rejected it. Most likely, if you're reading this book, you have probably joined the Java bandwagon in one way or another. And if you're like the rest of the world, you may not have had much choice in the decision. The choice may have been mandated by corporate culture or by a forced implementation forged by the IT Industry.

Like it or not, Java has evolved into the programming language defacto standard for the majority of the IT world. That includes the 64-bit RISC technology iSeries and especially WebSphere developers.

IBM has spent a great deal of time, effort, and money in the Java arena. In many cases, we have provided superior solutions by enhancing existing hardware and software. The iSeries is no exception, especially when it comes to such components as Java garbage collection.

This chapter introduces you to some key concepts and components of the Java and WebSphere environment as it pertains to the iSeries server.

2.1 Java execution environment

Whether you run Java programs on an iSeries server, IBM @server pSeries® server, or any other platform, there are many basic pieces of the programming puzzle that fit. This section covers the parts of the Java environment that relate to the iSeries server. This information is intended to help you to obtain a better understanding of how to tune and tweak your iSeries for Java applications that are running in WebSphere.

2.1.1 Java virtual machine implementation

After you install Java on your iSeries, along with some Java Development Kits (JDKs, 5722-JV1 for WebSphere on iSeries), you will have a fully functioning Java environment to play with.

The Java virtual machine (JVM) is the process or group of tasks that run WebSphere and your Java applications. The JVM on the iSeries is more tightly integrated into the operating system. It runs under the Technology Independent Machine Interface layer (TIMI), within System Licensed Internal Code (SLIC). With the JVM implemented within SLIC, threads are executed natively. That is that they are run or called directly, instead of through an operating system layer. There are other advantages on the iSeries such as the ability to make Java Native Interface (JNI) native method calls from ILE C or C++ programs (assuming they are setup as service programs *SRVPGM).

2.1.2 Java Runtime Environment

The Java Runtime Environment (JRE) is process space that allows you to run Java programs. This environment starts with the Run Java (RUNJVA) command, the Java command from the OS/400® command line, or the Java command from the Qshell Interpreter (STRQSH command, run using program QSHSH). Because Java is multi-threaded (multiple programs or processes can run at the same time), the JRE must run as a batch job, batch immediate (BCI) to be exact. Additional process run as well, including garbage collection.

Qshell is a process or job, similar to a terminal service. In it, you can run several UNIX®-like commands such as `ls` for a directory listing, `cd` to change the directory, or `ps` for displaying process status. Also, it's important to note that each command run with Qshell is actually another process or job that is being executed.

Another feature in V5R2 for Qshell is to redirect the output. You do this by changing the environment variable `QIBM_QSH_CMD_OUTPUT`, using the Add Environment Variable (ADDENVVAR) CL command as shown in the following example:

```
ADDENVVAR ENVVAR(QIBM_QSH_CMD_OUTPUT) VALUE('FILE=lsout.txt')
```

When you run the following CL command, the output is produced in the integrated file system (IFS) directory under the `lsout.txt` file:

```
STRQSH CMD('ls *.csv')
```

2.2 Java compilation environment

As with any Java program, the source code that you write must be compiled at some point. This is no different than most other languages. One feature of Java is that the source code that you write can be compiled on any operating system; Java is platform-independent. And when the source is compiled, you can move it from one system to the next, even while the

system is still running, without error. In some cases, such as with the iSeries, optimization options can make your code run more efficiently. The two options are:

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

2.2.1 Direct execution using Java transformation

With Java transformation, the iSeries pre-compiles your Java classes and Java Archive (JAR) files into persistent Java programs more suitable for the iSeries. These Java programs are hidden to the user, but are run underneath the covers when you use the Java command. The system checks to see if the pre-compiled Java program already exists. If so, it runs that program instead. This is known as *direct execution* (DE). DE provides the benefit of executing your Java bytecode directly with the machine interface (MI) code attached underneath the OS/400 layer.

With the use of the Java transformer, the pre-compiled code usually gives better performance in terms of computational efficiencies, especially when fully optimized. Keep in mind that the transformed Java class is hidden or transparent to the user. Only performance collection can see the programs. Also, the original Java class is accessible via the IFS and can still be run interpretively based on the correct parameters on the Java command.

Important: Since V4R5, the default for Java compilation on the iSeries does not use the Java transformer DE, but uses Just-in-time (JIT) compilation which is discussed next.

But how is DE used? DE programs are created by entering the Create Java Program (CRTJVAPGM) command. This command converts the .class and .jar files into direct execution (machine instruction) Java program objects. After the Java class is converted, it is permanent and will be reused on subsequent executions. You can use the Display Java Program (DSPJVAPGM) command to see if the hidden program exists. Incidentally, you can still use the CRTJVAPGM command to create non-DE programs. However, you must specify OPTIMIZE(*INTERPRET) as an optimization level for the CRTJVAPGM command. This bypasses the Java transformer process.

With the release of V5R1, you can perform DE on several JAR files at one time, assuming that they are all located in the same directory. This adds to performance by having the binding between all the classes more tightly integrated. This technique is known as Whole Program Optimization (WPO).

2.2.2 JIT compiler

The JIT compiler compiles your source code just in time for you to use it. It compiles the code the first time that you call a particular method within the code. The resulting class is a set of non-persistent machine instructions specifically for the platform on which it was compiled, the iSeries server. With JIT, the code you write is compiled immediately the first time. After the initial compilation, the response time of class execution should improve to the point where it is nearly as fast as DE.

One advantage to JIT is that it is more efficient in dynamic environments where classes are dynamically loaded quite often, but not used frequently. There is not the overhead that DE has for pre-compiling lesser used objects. Why should you compile them up front when you do not need them right away?

Another advantage of JIT is in regard to such products as WebSphere, where user-based class loaders are used. These class loaders load classes directly, without using the DE

programs associated with the classes. Therefore, even if you use the Java transformer to create your DE programs, WebSphere does not use them because of its user class loaders. Keep in mind you can override WebSphere's user class loaders so that the DE programs can be used. However in most cases the benefits are negligible.

And JIT has an advantage when doing performance analysis. Using DE programs (compiling class and JAR files with the CRTJVAPGM not in JIT mode) usually does not have entry and exit hooks for performance collection metrics. If you run them with the RUNJVA command using JIT mode by specify INTERPRET(*JIT), and the PROP((os400.enbpfrcol)), the *PGM created has the necessary hooks in it.

In summary, running Java programs using JIT is better. JIT takes longer to execute the first time but is as fast as DE after that. JIT is more efficient because only methods used are compiled. With JIT, your JAR files containing the code remain the same size. If you decide to use DE via the Java transformer, your initial startup time of the code is faster (in some cases ten times faster). However, it is not as efficient because all methods are optimized, and the JAR files are larger (in some cases ten times as large).

2.2.3 Compiling and running Java code

You can compile Java source code (class file) on the iSeries server by using the CRTJVAPGM command. You can do this for one class or several. You can also do this for a JAR or ZIP file, or a group of files. Depending on your needs, you may want to specify the optimization level differently.

You can specify *INTERPRET if you prefer to have your Java programs run on the fly. Otherwise, you can specify levels 10, 20, 30, or 40. Level 10 transforms the code with minimal compiler optimization, and level 40 transforms the code with the most optimization being performed. At level 40, such functions as call and instruction tracing are disabled, so level 40 is not good if want to do debugging.

You may also want to specify the Enable Performance Collection (ENBPFRCOL) parameter. It determines whether you want performance data to be collected for the object. The default is *NONE, but you can specify *FULL to collect detailed performance information.

To run Java code on the iSeries, you can use the RUNJVA command. There are several different parameters that can alter the amount of performance you receive. The two key parameters are the Optimization and Interpret parameters.

With the Optimization parameter, you can specify different levels of optimization, similar to the OPTIMIZE parameter on the CRTJVAPGM command. You can also specify *INTERPRET or *JIT. *INTERPRET makes the resulting Java program interpret the class byte code when invoked. This is great for debugging. If you specify *JIT for the Optimization parameter, no Java program is created with machine instructions if no program is associated with the class file. This increases performance over time as each method is invoked for the class.

With the Interpret parameter, you specify how the Java program interprets the code. Valid values are *OPTIMIZE, *YES, *NO, and *JIT. You can use *OPTIMIZE if you want your class file to run based on the Optimization parameter on either the RUNJVA command or CRTJVAPGM command. You can specify *JIT if you always want the Java class to run using JIT. We recommend that you use *JIT for all class files. Use *OPTIMIZE when performance issues are found for specific class files. In such a case, you may set the optimization level to 40. Or you can use level 30 if level 40 does not compile.

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 server. 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 Java Database Connectivity (JDBC) driver (part of JV1), the Java Toolbox classes installed with JC1, and the open version of the Java Toolbox (JTOpen).

2.3 Garbage collection

Every perfectly good application, whether it is tuned and tweaked to perfection, still results in extra useless information or “garbage” once it is done processing. This is similar to garbage that you may have in your house. For example, you may have empty boxes, banana peels, or newspapers that you throw out as soon as you are done using them. The information that you don't need anymore is thrown away, collected by the garbage man, and sent to be destroyed or recycled. As long as you define what is to be thrown into the garbage can, you can live and move freely within your home.

That's pretty much the same case when it comes to Java garbage collection. The garbage that is collected are *objects* that are no longer being used. The objects are marked as *garbage*, which means that they go into the garbage can. Eventually, on a scheduled basis, the “garbage man” runs around, picks up the unused objects, and throws them away. That sounds better than how we manage the garbage in our homes. What's even better is that on the iSeries, the garbage is picked up asynchronously, running in the background, out of your way. It is also picked up concurrently, working at the same time you're dropping garbage all around the house. Furthermore, the garbage collector (GC) finds objects or pieces of code that is no longer reachable, garbage that you didn't even know you had.

2.3.1 Basics in collecting garbage

Garbage collection for the iSeries differs from other platforms because it runs asynchronously. It is always lurking in the background, running whenever it needs to, and not getting in the way of your running application. There is no *stop and copy garbage collection*, where the application and JVM it runs in has to stop all threads or processes and clean up the lost and wandering objects. Most platforms use stop and copy garbage collection, but the iSeries does not. It uses *concurrent garbage collection*, which means your application continues merrily on its way all the while memory is cleaned.

For the iSeries, there are two primary parameters for garbage collection in regard to performance: the initial heap size and the maximum heap size. The *initial heap size* sets the initial size of how big you want the JVM's memory bucket or repository of objects to get before garbage collection takes place. Remember, as you throw away unused objects, the memory footprint has to increase to hold new usable objects. If this doesn't happen, then you cannot store a new object in memory until the old one is gone. Therefore, you must set the heap size at a specific size to clean up unused objects, and set a *maximum heap size* of the objects before and during cleanup.

When looking at the details of iSeries garbage collection, begin with the initial heap size. It should be called the *garbage collecting threshold*. It is the value that triggers the garbage collection to start running. When the JVM allocates memory and reaches this threshold value, the GC begins running until the amount of memory allocated is cleaned up. Incidentally, the maximum heap size value on the iSeries should almost always be set to *NOMAX.

2.3.2 Garbage collection pitfalls

One of the common myths about garbage collection is that it relieves the programmer from all memory management responsibility. This is not entirely true. There are certain actions that the programmer must take to prevent dangling objects that can cause memory leaks and create problems while your application swims through its memory pool. Without proper attention to the use of objects in your application, the pool soon fills up with “dead” objects. There may also be programmers who take on too much in the memory management arena. They spend valuable time trying to make sure that all objects that are no longer needed are done, dead, and destroyed.

Where is the balance? Since memory is freed only after all references to the objects are removed, developers must use care to ensure that this is done. You may find yourself writing code that leaves objects out when the method has finished executing. A good way to prevent this is to set objects or variables in your method to *null* in your final clause or at the end of your method. There may also be issues where you load objects into a cache, but you never clean the cache. These unused cached objects are never cleaned by garbage collector, because they are still referenced by the cache.

Other such pitfalls of garbage collection include finding the right heap size. What should you set the initial heap size to? If it is set too low, then garbage cleanup is faster, because there is less garbage (smaller garbage can), but more frequent. But if you set the heap size too high, then garbage cleanup takes longer (larger garbage can), but it happens less frequently. The bottom line is to find the point at which the frequency of garbage collection is low enough so it does not cause any overhead, yet often enough that the heap size does not grow too large.

2.4 Profiling (PRFDTA)

In some cases, you may want to look at profiling your programs to ensure that they are running in the most optimized way. One of the best ways to do this is to collect profile data on your programs.

2.4.1 Profiling on the iSeries

Program profiling is an advanced optimization technique to reorder procedures and code within the procedures based on statistical data (profiling data). You do this by using the Profiling Data (PRFDTA) parameter on the CRTJVAPGM, Change Java Program (CHGJVAPGM), Change Program (CHGPGM), and Change Service Program (CHGSRVPGM) commands.

2.4.2 Collecting profiling data

A Java program is enabled to collect profiling data using option PRFDTA(*COL) on the CRTJVAPGM or CHGJVAPGM command

The Start Program Profiling (STRPGMPRF) command starts to collect profiling information from ILE programs or service programs. These programs are enabled to collect profiling data when you specify PRFDTA(*COL) on the CHGPGM or CHGSRVPGM command, or when a compiler creates modules, or you specify PRFDTA(*COL) on the Change Module (CHGMOD) CL command. All active programs that are compiled or changed with this option have profiling information updated until you enter the End Program Profiling (ENDPGMPRF) command.

The profiling information is added to the existing profiling information. If you do not want this, you can clear the profiling data by specifying PRFDTA(*CLR) on the CHGPGM or CHGSRVPGM command.

The ENDPGMPRF command ends collection of program profiling data for programs or service programs that were enabled to collect profiling data using the PRFDTA(*COL) option on the CHGPGM or CHGSRVPGM CL command, or when you created the modules using the CHGMOD CL command.

2.5 IBM Toolbox for Java

Every good developer needs a good set of tools to write code. On the iSeries, there is no exception. In fact, the iSeries has a whole toolbox of tools. The toolbox is loaded with Java classes and tools that allow you, the programmer, to more efficiently access iSeries data and resources. They help you to directly link to iSeries data for Java applications, client server applications, applets, servlets, etc. You can even use the toolbox within an application, with your application accessing the Java class tools directly.

A key feature to the IBM Toolbox for Java is that it uses the iSeries Host servers as access points to the system. Each server has its own job and sends and receives data via separate socket connections. This is similar to the how the Client Access/400 application programming interfaces (APIs) work. There is also a specific JDBC driver for accessing DB2 data on the iSeries. Each set of Java classes is grouped into packages. For example, the commtrace package is grouped into com.ibm.as400.commtrace.

The different groupings of tools found in the toolbox are:

- ▶ Java Toolbox packages
- ▶ Graphical Toolbox
- ▶ Extensible Markup Language (XML) components
- ▶ Proxy support
- ▶ Security
- ▶ System properties

2.6 WebSphere execution environment

Now that you know the basic details about the Java environment on the iSeries, let's discuss some basics about the WebSphere environment. What is WebSphere? In terms of this book, it is the WebSphere Application Server. It is the part of the dynamic Web-puzzle that builds the content for you Web pages. On the front end, you have a Web server such as the IBM HTTP Server (powered by Apache). On the back end, you have a database, such as DB2 Universal Database (UDB). And in the middle, you have WebSphere Application Server.

With WebSphere as the dynamic middle-man, serving up Web content from business data, there are lots of considerations to make, especially in regard to performance. To ensure that everything is running smoothly, you must address and assemble many parts correctly.

The key areas to consider in regard to WebSphere are:

- ▶ Administration
- ▶ Application development
- ▶ Web services
- ▶ Security
- ▶ Troubleshooting

2.6.1 Administration

With all the complexities involved with running an application in WebSphere or any application server for that matter, some heavy duty administration tools must be available. And WebSphere on the iSeries is no exception. Furthermore, tools must be available to help evaluate performance issues, configuration management, and resource availability. WebSphere has those tools too.

Whether your application environment is setup with a simple Web server, application server, and database, or it is a highly complex setup using clustering and load balancing, WebSphere has the right tools to manage all aspects of the application.

Administrative console

WebSphere has a highly useful browser-based administrative console graphical user interface (GUI) tool that allows you to deploy your Enterprise Archive (EAR) files, configure your server, setup logging, or create your data source. It enables you to perform all aspects in regard to administration.

Configuration management

Configuring WebSphere is no different than for any other application server. Based on Java 2 Enterprise Edition (J2EE) principles and Web application standards, you can configure your application to access various resources such as a Java Messaging Service (JMS) server or database, or even other applications. Configuration information for your application (or your server for that matter) is setup in various XML files.

Performance

WebSphere Application Server is shipped with a stand-alone performance monitoring application called *Tivoli® Performance Viewer*. This tool is powerful when it comes to tracking down performance problems. Also various logs are created that you can use to determine where the performance problem is. Keep in mind that the iSeries has a plethora of performance gathering tools. The data collection options are endless.

Advanced topologies

WebSphere Application Server ships with a single application server topology. WebSphere Application Server Network Deployment allows you to configure multiple machine environments in which applications are deployed on several application servers. A more complex environment may also include discussions about firewalls, Web server separation techniques, horizontal and vertical scaling, and multiple WebSphere Application Server cells.

High availability

With WebSphere Application Server Network Deployment, you can use clustering to deploy applications on multiple machines or logical partitions. Clustering provides failover support and enables workload management.

Administration reference

This topic provides reference information about WebSphere Application Server administration. It includes information about file and directory structure, user profiles and authorities, product scripts and commands, and port usage.

2.6.2 Application development

WebSphere Application Server 5.1 supports the J2EE 1.4 specification for application components. An enterprise application is composed of application modules that work

together to perform a business logic function. A J2EE-compliant application can contain any of these components:

- ▶ Enterprise beans
- ▶ Servlets, JavaServer Pages (JSP) files and other Web components
- ▶ Resource adapter (connector) implementations
- ▶ Application clients
- ▶ Other supporting classes and files

Development tools

You can use any application development tool that supports J2EE application development. One tool of choice WebSphere Development Studio Client for iSeries. The key benefit to WebSphere Development Studio Client is that it has a built-in WebSphere Test Environment (WTE). WTE is not a stripped down WebSphere Application Server, but a complete version of the product running on your workstation.

With WebSphere Development Studio Client in place, you can begin creating your application. You can develop all that you need using WebSphere Development Studio Client, whether it is HTML, JSPs, XML, Servlets, EJBs, or Web services. You are done with the development, you assemble your Web code into an EAR file to perform testing with WTE. Finally, when everything is working perfectly, you can deploy your EAR file, either with WebSphere Development Studio Client or an external tool such as Ant.

2.6.3 Security

WebSphere Application Server security is built on a layered security architecture, as shown in Figure 2-1. Each layer offers security protection.

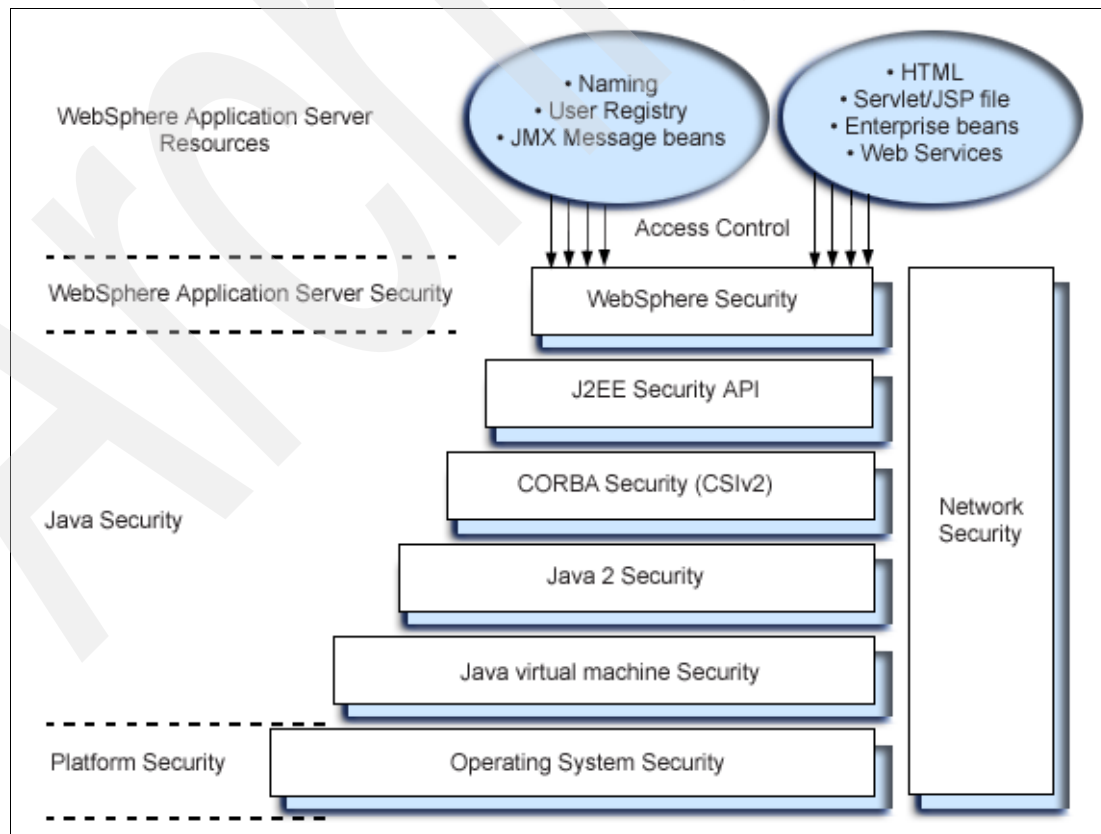


Figure 2-1 WebSphere security model

The building blocks of WebSphere security are:

- ▶ **Operating system security**

The security infrastructure of the underlying operating system provides certain security services to the WebSphere Security Application. This includes file system security support to secure sensitive files in WebSphere product installation. You can configure the product to obtain authentication information directly from the operating system user registry.

- ▶ **Network security**

These layers provide transport-level authentication as well as message integrity and encryption. You can configure WebSphere Application Server inter-server communications to use Secure Socket Layer (SSL) and HTTP Secured (HTTPS). In addition, you may use IP Security and Virtual Private Network (VPN) for added message protection.

- ▶ **JVM 1.4.2**

The JVM security model provides a layer of security above the operating system layer.

- ▶ **Java 2 security**

The Java 2 Security model offers fine-grained access control to system resources including file system, system property, socket connection, threading, class loading, and so on. Application code must explicitly grant the required permission to access a protected resource.

- ▶ **CORBA security**

Any calls made among secure Object Request Brokers (ORBs) are invoked over the Common Security Interoperability Version 2 (CSIv2) security protocol. This protocol sets up the security context and the necessary quality of protection. After the session is established, the call is passed up to the enterprise bean layer. WebSphere Application Server supports the Secure Authentication Service (SAS) security protocol.

- ▶ **J2EE security**

The security collaborator enforces J2EE-based security policies and supports J2EE security APIs.

- ▶ **WebSphere security**

WebSphere security enforces security policies and services in a unified manner on access to Web resources, enterprise beans, and Java Management Extensions (JMX) administrative resources. It consists of WebSphere security technologies and features to support the needs of a secure enterprise environment.

2.6.4 Web services

Web services are self-contained, modular applications that you can describe, publish, locate, and invoke over a network. They reflect a new, service-oriented approach to programming. This approach is based on the idea of building applications by discovering and implementing network services or by invoking available applications to accomplish some task. This approach is independent of specific programming languages or operating systems.

Web services deliver interoperability. They provide the ability for components created in different programming languages to work together as though they were created using the same language. Web services rely on existing transport technologies, such as HTTP or JMS, and standard data encoding techniques, such as XML, to invoke the implementation.

Performance tuning methodology

This chapter introduces the basics of performance analysis methodology, as it relates to the iSeries WebSphere Application Server environment. You should be familiar generally with the practice of performance management since this is not a complete discussion about performance methodology.

In a live production environment, any number of factors may influence the performance of the system and your applications. Some of these factors may include improperly configured main storage, unexpected workload spikes, or the introduction of a new application. Since performance issues can be derived from any number of sources, it is paramount to have an overarching strategy to manage and resolve performance-related concerns.

This chapter briefly reviews the steps that you can use to derive a performance management strategy. In addition, it summarizes the key tools that you may use to measure performance. Beyond this chapter and Chapter 4, “Tools for determining performance problems” on page 27, where you learn about each of the tools, you should also read the individual documentation for each tool to learn how to apply what you learn in this redbook.

3.1 Setting performance objectives

The first step in managing system performance is to set measurable objectives. Begin by setting goals that match the demands of your business and by identifying areas of the system where performance improvements can have a positive effect. Implementing a performance strategy is an iterative process. It begins with defining your performance objectives and then repeating a series of tasks until you accomplish your stated goals.

The following repeatable tasks make up a performance strategy:

1. Set performance objectives.
 - 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 you can measure.
 - 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 your stated 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 identify 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 your system whenever guidelines are not met.
 - When tuning your system, make only one change at a time. If you make single changes, you can then determine the effect of each specific change. By prioritizing the changes, you can incrementally make changes and evaluate the performance measurements 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 10, “Scaling and capacity planning for WebSphere applications” on page 345.

3.2 General tuning process and performance strategy

The most effective tuning strategy must commence at the operating system level and proceed outward from there to more detailed WebSphere Application Server and application-level tuning options. To achieve optimal performance and stability in a WebSphere Application Server or Java application deployment, you should follow these steps:

1. Prerequisite: Ensure that you have adequate system resources.

Before you start thinking about tuning your system, you must be confident that you have a system that is up to the task at hand. Some iSeries server models are inadequate for Java and WebSphere workloads. For example, some older AS/400 models may have high central processing workload (CPW) ratings. But when it comes to executing Java, central processing unit (CPU) MHz and the size of L2 cache are more important. This is covered in greater detail in Chapter 10, “Scaling and capacity planning for WebSphere applications” on page 345.

2. Understand the environment.

In the tuning process, you must understand what you are tuning. Java applications behave differently than traditional RPG and COBOL applications. They are multithreaded by nature, which breaks several of the traditional rules of thumb used for operating system tuning for host applications. Chapter 2, “Java and WebSphere execution environment on iSeries servers” on page 11, covers the Java environment.

3. Optimize the operating system for your environment.

When 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. Often a simple setting change can produce favorable results. You can learn more about tuning the operating system in Chapter 5, “Tuning iSeries for a WebSphere or Java environment” on page 165.

4. Tune the WebSphere Application Server.

The next relatively easy opportunity for performance gains is within the WebSphere Application Server itself. There are a large number of “knobs” that you can turn to gain extra performance from your system. Learn more about the knobs in WebSphere Application Server, Base version, in Chapter 8, “Tuning WebSphere Application Server Base” on page 231.

5. Tune your application

The next and often the most costly step is to change the application code. Understanding the coding practices that adversely impact performance is important. Chapter 9, “Java and WebSphere application design” on page 285, presents the best practices for Java and WebSphere development.

Identifying where poorly performing code is located can be difficult. Consider performing some profiling during the development cycle to more easily locate the portions of your application that are not performing as well as you want. Chapter 4, “Tools for determining performance problems” on page 27, and Chapter 9, “Java and WebSphere application design” on page 285, include information about profiling tools that you can use while developing your application to measure performance.

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

We recommend that application developers consider load or stress testing your application several times throughout the development cycle. Then you can identify performance problems much earlier in the development life cycle and apply corrective action to the code before bad practices spread throughout the application. Chapter 4, “Tools for determining performance problems” on page 27, outlines stress testing tools.

7. Scale the application.

If all your tuning efforts fail, consider scaling the application by using clustering or a virtualized configuration. Consider this as a last resort, because there is, in fact, no guarantee that your application will scale gracefully. Additional scalability testing is required prior to scaling your production environment. For more information about scaling see Chapter 10, “Scaling and capacity planning for WebSphere applications” on page 345.

3.3 Performance problem determination process

We strongly recommend that you use a formal problem determination process, since it reduces the amount of time needed to identify and resolve performance problems. Your objective should always be to resolve the production issues with a minimum measurement effort. Such a process should roughly follow this path:

1. Initiate with problem definition

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 back track and validate prior decisions, or move easily into an alternate investigation path.

3. Explore

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 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. Collect information

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. Perform resolution and closure

Given that you followed the process presented earlier (or something similar), then almost by definition, this step requires minimal effort. This does not mean that additional work is unnecessary. Rather it means that, for the specific exercise considered, a resolution is documented and the problem initially defined has been resolved.

While executing the performance problem determination process, you will likely uncover other avenues where further resource and task allocation could yield significant improvements. Therefore, you can consider the resolution process iterative in nature.

The following section summarizes the operating system, WebSphere, Java, and application level tools that can assist you in isolating and resolving performance problems.

3.4 Summary of key performance measurement tools

Depending on the type of problem and the depth of measurement, you can use one or more tools to locate the source of the issue. The tools listed in Table 3-1 are well suited for solving the vast majority of common performance issues that you are likely to encounter.

Table 3-1 Performance analysis tools

| Phase | Data collected by | Data analyzed by |
|---------------------------|---|--|
| System analysis | <ol style="list-style-type: none">1. Management Central2. Collection Services3. WRKSYSSTS command4. WRKSYSACT command5. WRKACTJOB command6. WRKDSKSTS command7. iDoctor tools | <ol style="list-style-type: none">1. Management Central2. Performance Tools for iSeries3. Interactive user4. Interactive user5. Interactive user6. Interactive user7. Interactive user |
| WebSphere analysis | <ol style="list-style-type: none">1. WebSphere Studio2. Performance Monitoring Infrastructure3. Performance Advisors4. Log Analyzer5. Thread Analyzer | <ol style="list-style-type: none">1. Profiling and Logging perspective or developer2. Java, Web, or JMX client or Tivoli Performance Viewer3. Interactive user4. Interactive user5. Interactive user |
| Java/application analysis | <ol style="list-style-type: none">1. DMPJVM command2. ANZJVM command3. HEAPANA command4. STRSST GC tools | <ol style="list-style-type: none">1. Interactive user2. Interactive user3. Interactive user4. Interactive user |
| HTTP analysis | <ol style="list-style-type: none">1. Server logging2. HTTP tracing3. Collection Services4. Communications trace | <ol style="list-style-type: none">1. Interactive user2. Interactive user3. Performance Tools for iSeries4. Interactive user |
| Database analysis | <ol style="list-style-type: none">1. DBMonitor (DBMON)2. SQL Visual Explain | <ol style="list-style-type: none">1. iSeries Navigator or Interactive User2. iSeries Navigator or Interactive User |

Chapter 4, “Tools for determining performance problems” on page 27, examines these tools in greater detail.

Archived

Tools for determining performance problems

This chapter describes the tools that you should use to determine the reason for the performance problem. It introduces the use of:

- ▶ System wide performance tools
- ▶ Java performance tools
- ▶ WebSphere performance tools
- ▶ HTTP performance tools
- ▶ Stress testing tools

The chapters that follow describe the detailed use of these tools to assist with performance-related issues.

4.1 System-level tools

Use the following tools to determine the overall use of system resources either in real time or reactively, such as analyzing previously collected performance data.

4.2 System-level tools for real time use

System-level tools refer to the tools that are capable of measuring system resources, such as central processing unit (CPU) usage, memory pool page faulting, disk utilization, and so on. The tools that are most often used for system wide performance monitoring are:

- ▶ Work with System Status (WRKSYSSTS) command
- ▶ Work with Disk Status (WRKDSKSTS) command
- ▶ Work with Active Jobs (WRKACTJOB) command
- ▶ Management Central

One set of tools that you should not forget is System Service Tools (SST). SST provide information about the hardware status on your iSeries server. Using 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.

4.2.1 WRKSYSSTS command

The Work with System Status display (Figure 4-1) shows a group of statistics that depict the current status of the system. The display is actually two groups of information on one screen. The upper half shows statistics about the status of the system, while the lower half is used to both observe and adjust the memory pool sizes and activity levels. The example in Figure 4-1 of the WRKSYSSTS display shows the advanced assistance level. You may also turn on the expert cache by using the WRKSYSSTS command.

| Work with System Status | | | | | | | | | | SYSTEMA |
|-----------------------------|---------|----------|----------------------------------|-------------|-------------|-------|-------|------|--|-------------------|
| | | | | | | | | | | 07/07/04 12:40:45 |
| % CPU used | : | 71.8 | System ASP | : | 175.4 G | | | | | |
| % DB capability | : | 11.9 | % system ASP used | : | 25.6190 | | | | | |
| Elapsed time | : | 00:05:21 | Total aux stg | : | 175.4 G | | | | | |
| Jobs in system | : | 464 | Current unprotect used | : | 18369 M | | | | | |
| % perm addresses | : | .007 | Maximum unprotect | : | 39388 M | | | | | |
| % temp addresses | : | .010 | | | | | | | | |
| Sys | Pool | Reserved | Max | ----DB---- | --Non-DB--- | Act- | Wait- | Act- | | |
| Pool | Size M | Size M | Act | Fault Pages | Fault Pages | Wait | Inel | Inel | | |
| 1 | 435.67 | 237.98 | +++++ | .0 .0 | .1 .1 | 231.8 | .0 | .0 | | |
| 2 | 6406.74 | 25.59 | 621 | .0 .0 | 2.6 9.5 | 2047 | .0 | .0 | | |
| 3 | 528.76 | .01 | 150 | .0 .0 | .2 .2 | 5.9 | .0 | .0 | | |
| 4 | .25 | .00 | 1 | .0 .0 | .0 .0 | .0 | .0 | .0 | | |
| 5 | 100.71 | .21 | 25 | .0 .0 | .0 .0 | 240.4 | .0 | .0 | | |
| 6 | 2600.00 | .79 | 85 | .1 .9 | 12.4 103.5 | 14629 | 26.8 | .0 | | |
| | | | | | | | | | | Bottom |
| ===> | | | | | | | | | | |
| F21=Select assistance level | | | | | | | | | | |

Figure 4-1 WRKSYSSTS display

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

- ▶ The 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

The lower half of the WRKYSSTS display shows:

- ▶ The sizes of the memory pools on the system
- ▶ The activity level settings per pool basis
- ▶ The faulting and paging rates for each pool for
 - Database objects
 - Non-database objects like program code, work areas, and so on
- ▶ Thread state transitions for each pool

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

4.2.2 WRKDSKSTS command

You use the WRKDSKSTS command to observe how the disks are performing. Figure 4-2 shows an example of Work with Disk Status display.

| Work with Disk Status | | | | | | | | | | SYSTEMA |
|---|------|----------|--------|---------|------------------|----------|-----------|----------|-----------|-------------------|
| | | | | | | | | | | 07/07/04 12:42:31 |
| Elapsed time: 00:00:08 | | | | | | | | | | |
| Unit | Type | Size (M) | % Used | I/O Rqs | Request Size (K) | Read Rqs | Write Rqs | Read (K) | Write (K) | % Busy |
| 1 | 6718 | 13161 | 25.7 | 20.5 | 4.0 | .1 | 20.4 | 4.0 | 4.0 | 5 |
| 2 | 6718 | 13161 | 25.6 | 14.6 | 4.2 | .1 | 14.4 | 4.0 | 4.2 | 5 |
| 3 | 6718 | 17548 | 25.6 | 29.5 | 5.1 | .6 | 28.9 | 32.8 | 4.5 | 0 |
| 4 | 6718 | 13161 | 25.6 | 10.2 | 4.8 | .3 | 9.8 | 10.6 | 4.5 | 5 |
| 5 | 6718 | 13161 | 25.6 | 29.7 | 4.3 | .3 | 29.3 | 22.6 | 4.0 | 5 |
| 6 | 6718 | 17548 | 25.6 | 25.2 | 5.3 | .1 | 25.0 | 4.0 | 5.3 | 0 |
| 7 | 6718 | 17548 | 25.6 | 18.5 | 4.5 | .2 | 18.2 | 12.0 | 4.4 | 5 |
| 8 | 6718 | 17548 | 25.6 | 60.4 | 4.2 | .3 | 60.0 | 4.0 | 4.2 | 5 |
| 9 | 6718 | 17548 | 25.6 | 43.7 | 5.5 | .2 | 43.4 | 6.0 | 5.5 | 5 |
| 10 | 6718 | 17548 | 25.6 | 39.8 | 4.3 | .3 | 39.4 | 22.6 | 4.1 | 0 |
| 11 | 6718 | 17548 | 25.6 | 32.0 | 4.6 | .6 | 31.4 | 4.0 | 4.6 | 11 |
| | | | | | | | | | | Bottom |
| Command | | | | | | | | | | |
| ===> | | | | | | | | | | |
| F3=Exit F5=Refresh F12=Cancel F24=More keys | | | | | | | | | | |

When you view the Work with Disk Status display, observe the % Busy column. 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 may 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, increase the number of disk actuators.

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

When you press F11 on the display shown in Figure 4-2, you see information about the status of the individual disk units at a glance. Figure 4-3 shows an example of this information.

| Work with Disk Status | | | | | SYSTEMA |
|---|-----|------|--------|-------------|-------------------|
| Elapsed time: 00:00:00 | | | | | 07/07/04 12:42:57 |
| --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 | | | | | Bottom |
| ===> | | | | | |
| F11=Display disk statistics F16=Work with system status F24=More keys | | | | | |

Figure 4-3 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:** This status indicates that 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:** This status 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:** This status 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:** This status indicates that a unit has failed. If another unit fails, data could be lost.

- ▶ **HDW FAIL:** This status 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:** This status indicates that this unit is part of a disk unit subsystem that has device parity protection. The unit is not ready to perform input/output (I/O) operations.
- ▶ **PWR LOSS:** This status indicates that this unit is part of a disk unit subsystem that has device parity protection. This unit has lost power.
- ▶ **REBUILD:** This status indicates that the data on this storage unit is being rebuilt from another storage unit in the disk unit subsystem.
- ▶ **RESUME:** This status indicates that the unit is part of a mirrored ASP, and mirroring is in the process of being resumed on this unit.
- ▶ **RW PROT:** This status 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:** This status indicates that the unit is part of a mirrored ASP, and mirroring is suspended on this unit.
- ▶ **UNKNOWN:** This status indicates that the protection status of the storage unit with device parity protection is not known to the system.
- ▶ **UNPROT:** This status indicates that the unit is operational, but another unit in the disk unit subsystem has failed or is being rebuilt.
- ▶ **WRT PROT:** This status 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.

4.2.3 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.

You press the F11 key to toggle the display between various sets of information shown on the screen. Then you press F16 to arrange data according to any metrics you want. You can see an example of this in Figure 4-7 on page 33, 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. Press F11 continuously until you see the Threads column.
3. Move the cursor onto the Threads column.
4. Press F16 to rearrange the jobs according to the column.

Figure 4-4 through Figure 4-7 on page 33 show the displays that you see when you perform this procedure. Figure 4-4 shows the Work with Active Jobs displays that you see after you enter the WRKACTJOB command.

| Work with Active Jobs | | | | | | | ASM05 |
|-------------------------------|---------------|---------------|----------|--------------|--------------|--------|-------------------|
| | | | | | | | 07/07/04 12:45:03 |
| CPU %: | 71.5 | Elapsed time: | 00:12:54 | Active jobs: | 287 | | |
| 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 | |
| | QEJBAS51 | QSYS | SBS | .0 | | DEQW | |
| | TRADE51 | QEJBAS51 | BCH | 53.2 | * -COMMIT | JVAW | |
| | QHTTSPVR | QSYS | SBS | .0 | | DEQW | |
| | ADMIN | QTMHHTTP | BCH | .0 | PGM-QZHBHTTP | SIGW | |
| | ADMIN | QTMHHTTP | BCI | .0 | PGM-QZSRLOG | SIGW | |
| | ADMIN | QTMHHTTP | BCI | .0 | PGM-QZSRHTTP | SIGW | |
| | ADMIN | QTMHHTTP | BCI | .0 | PGM-QYUNLANG | TIMW | |
| | ADMIN | QTMHHTTP | BCI | .0 | PGM-QZSRCGI | TIMW | |
| | | | | | | | More... |
| ====> | | | | | | | |
| F21=Display instructions/keys | | | | | | | |

Figure 4-4 Basic WRKACTJOB display

Then you press F11 and see the elapsed data (and the pool information) as shown in Figure 4-5.

| Work with Active Jobs | | | | | | | | | | ASM05 |
|-------------------------------|---------------|---------------|----------|--------------|--------|-----|-----|-------|-------|-------------------|
| | | | | | | | | | | 07/07/04 12:45:03 |
| CPU %: | 71.5 | Elapsed time: | 00:12:54 | Active jobs: | 287 | | | | | |
| Opt | Subsystem/Job | Type | Pool | Pty | CPU | Int | Rsp | AuxIO | CPU % | |
| | QBATCH | SBS | 2 | 0 | .0 | | | 0 | .0 | |
| | QCMN | SBS | 2 | 0 | .1 | | | 0 | .0 | |
| | QCTL | SBS | 2 | 0 | .1 | | | 0 | .0 | |
| | QSYSSCD | BCH | 2 | 10 | .1 | | | 0 | .0 | |
| | QEJBAS51 | SBS | 2 | 0 | .2 | | | 0 | .0 | |
| | TRADE51 | BCH | 6 | 20+ | 2456.9 | | | 0 | 53.2 | |
| | QHTTSPVR | SBS | 2 | 0 | .0 | | | 0 | .0 | |
| | ADMIN | BCH | 2 | 25 | 3.6 | | | 0 | .0 | |
| | ADMIN | BCI | 2 | 25 | 140.0 | | | 0 | .0 | |
| | ADMIN | BCI | 2 | 25 | 212.6 | | | 0 | .0 | |
| | ADMIN | BCI | 2 | 25 | .3 | | | 0 | .0 | |
| | ADMIN | BCI | 2 | 25 | .2 | | | 0 | .0 | |
| | | | | | | | | | | More... |
| ====> | | | | | | | | | | |
| F21=Display instructions/keys | | | | | | | | | | |

Figure 4-5 WRKACTJOB with elapsed data

When you press F11 a second time, you see the thread information as shown in Figure 4-6.

| Work with Active Jobs | | | | | | | ASM05 |
|-------------------------------|---------------|----------|------------------------|------|------------------|---------|-------------------|
| CPU %: 71.5 | | | Elapsed time: 00:12:54 | | Active jobs: 287 | | 07/07/04 12:45:03 |
| Opt | Subsystem/Job | User | Number | Type | CPU % | Threads | |
| | QBATCH | QSYS | 003195 | SBS | .0 | 1 | |
| | QCMN | QSYS | 003197 | SBS | .0 | 1 | |
| | QCTL | QSYS | 003156 | SBS | .0 | 1 | |
| | QSYSSCD | QPGMR | 003192 | BCH | .0 | 1 | |
| | QEJBAS51 | QSYS | 003306 | SBS | .0 | 1 | |
| | TRADE51 | QEJB5VR | 005899 | BCH | 53.2 | 73 | |
| | QHTT5VR | QSYS | 003308 | SBS | .0 | 1 | |
| | ADMIN | QTMHHTTP | 003336 | BCH | .0 | 1 | |
| | ADMIN | QTMHHTTP | 003345 | BCI | .0 | 1 | |
| | ADMIN | QTMHHTTP | 003348 | BCI | .0 | 32 | |
| | ADMIN | QTMHHTTP | 003577 | BCI | .0 | 1 | |
| | ADMIN | QTMHHTTP | 004229 | BCI | .0 | 1 | |
| | | | | | | | More... |
| ===> | | | | | | | |
| F21=Display instructions/keys | | | | | | | |

Figure 4-6 The WRKACTJOB with thread data

Now you move the cursor to the Threads column and press F16 to arrange the jobs on display according to the number of threads they have as shown in Figure 4-7.

| Work with Active Jobs | | | | | | | ASM05 |
|-------------------------------|---------------|----------|------------------------|------|------------------|---------|-------------------|
| CPU %: 71.2 | | | Elapsed time: 00:14:50 | | Active jobs: 287 | | 07/07/04 12:46:59 |
| Opt | Subsystem/Job | User | Number | Type | CPU % | Threads | |
| | QYPSJSVR | QYPSJSVR | 003246 | BCH | .1 | 240 | |
| | TRADE51 | QEJB5VR | 005899 | BCH | 53.1 | 73 | |
| | ADMIN | QTMHHTTP | 003348 | BCI | .0 | 32 | |
| | NODEAGENT | QEJB5VR | 005536 | BCH | .1 | 29 | |
| | QYPSFRCOL | QSYS | 003300 | BCH | .0 | 23 | |
| | QYPSSRV | QSYS | 006026 | BCH | .1 | 16 | |
| | AMQZLAAO | QMOM | 005551 | BCH | .0 | 11 | |
| | QDBFSTCCOL | QSYS | 003153 | SYS | .0 | 9 | |
| | QDIRSRV | QDIRSRV | 003239 | BCH | .2 | 7 | |
| | AMQZLAAO | QMOM | 005558 | BCH | .0 | 5 | |
| | AMQZXMAO | QMOM | 005545 | BCH | .0 | 4 | |
| | AMQRMPPA | QMOM | 005556 | BCH | .0 | 4 | |
| | RUNMQCHI | QMOM | 005550 | BCH | .0 | 3 | |
| | | | | | | | More... |
| ===> | | | | | | | |
| F21=Display instructions/keys | | | | | | | |

Figure 4-7 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. When you press F7 on any WRKACTJOB display, you see the display shown in Figure 4-8. On this display, you type the string that you are looking for, specify the column in which you want to find the string, and press Enter.

| Work with Active Jobs | | | | AS05 | |
|--|-------------|-------------------------------|----|---------------|---------|
| | | | | | |
| 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 | JVAW |
| TEAM20MNTR | 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 |
| ADMIN | QTMHHTTP | BCI | .0 | PGM-QYUNLANG | TIMW |
| ADMIN | QTMHHTTP | BCI | .0 | PGM-QYUNLANG | TIMW |
| | | | | | More... |
| ===> | | | | | |
| F21=Display instructions/keys | | | | | |

Figure 4-8 WRKACTJOB with the Find a string function

The display shown in Figure 4-9 shows the result of searching for the RUN string in the job Status column that was created in Figure 4-8.

| Work with Active Jobs | | | | | | AS05 | |
|-------------------------------|---------------|---------------|----------|--------------|---------------|----------|----------|
| | | | | | | 07/07/04 | 12:48:58 |
| CPU %: | 22.9 | Elapsed time: | 00:10:56 | Active jobs: | 341 | | |
| Opt | Subsystem/Job | User | Type | CPU % | Function | Status | |
| | PETRIS | FAULT | 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 4-9 The WRKACTJOB display with Find a string results

4.2.4 WRKSYSACT command

The Work with System Activity (WRKSYSACT) display allows you to view performance data from a 5250 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. 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 (LPP).
- ▶ Only one user at a time may use the WRKSYSACT command.

Figure 4-10 shows an example of the WRKSYSACT display.

| Work with System Activity | | | | | | | | | | SYSTEMA | |
|---|--|----------|--|----------------------------|--|----------|--|-------|--|-----------------------------------|--|
| | | | | | | | | | | 07/07/04 12:48:52 | |
| Automatic refresh in seconds | | | | | | | | | | 5 | |
| Elapsed time | | 00:00:35 | | Average CPU util | | 66.8 | | | | | |
| Number of CPUs | | 2 | | Maximum CPU util | | 72.5 | | | | | |
| Overall DB CPU util | | 10.7 | | Minimum CPU util | | 61.0 | | | | | |
| | | | | | | | | | | Current processing capacity: 2.00 | |
| Type options, press Enter. | | | | | | | | | | | |
| 1=Monitor job 5=Work with job | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | Total | | Total | | DB | |
| Job or | | | | | | CPU | | Sync | | Async | |
| Opt Task | | User | | Number Thread | | Pty Util | | I/O | | I/O | |
| | | | | | | | | | | CPU | |
| TRADE51 | | QEJBSVR | | 005899 000003BF | | 26 11.0 | | 317 | | 247 | |
| TRADE51 | | QEJBSVR | | 005899 000003C0 | | 26 8.6 | | 396 | | 345 | |
| TRADE51 | | QEJBSVR | | 005899 000003C1 | | 26 7.1 | | 254 | | 176 | |
| TRADE51 | | QEJBSVR | | 005899 000003C2 | | 26 5.3 | | 285 | | 258 | |
| TRADE51 | | QEJBSVR | | 005899 000003C3 | | 26 3.9 | | 127 | | 127 | |
| TRADE51 | | QEJBSVR | | 005899 000003C4 | | 26 3.0 | | 116 | | 115 | |
| TRADE51 | | QEJBSVR | | 005899 000003C5 | | 26 2.2 | | 64 | | 46 | |
| QSQRVR | | QUSER | | 005965 00000085 | | 20 1.8 | | 1 | | 1463 | |
| | | | | | | | | | | 1.9 | |
| More... | | | | | | | | | | | |
| F3=Exit F10=Update list F11=View 2 F12=Cancel F19=Automatic refresh | | | | | | | | | | | |
| F24=More keys | | | | | | | | | | | |

Figure 4-10 WRKSYSACT display

You can press the F11 key to toggle between the four different views of this display in a rotating order. You 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
- ▶ Total waiting time

4.2.5 Management Central

You can use Management Central's Collection Services to collect performance data as described in 4.3.1, "Collecting performance data" on page 93. You may use the Performance Tools for iSeries LPP (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. This in turn gives you the best performance from your system. You can customize your data collections so you collect only the data that 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 4-11 shows an example of a Management Central CPU monitor.

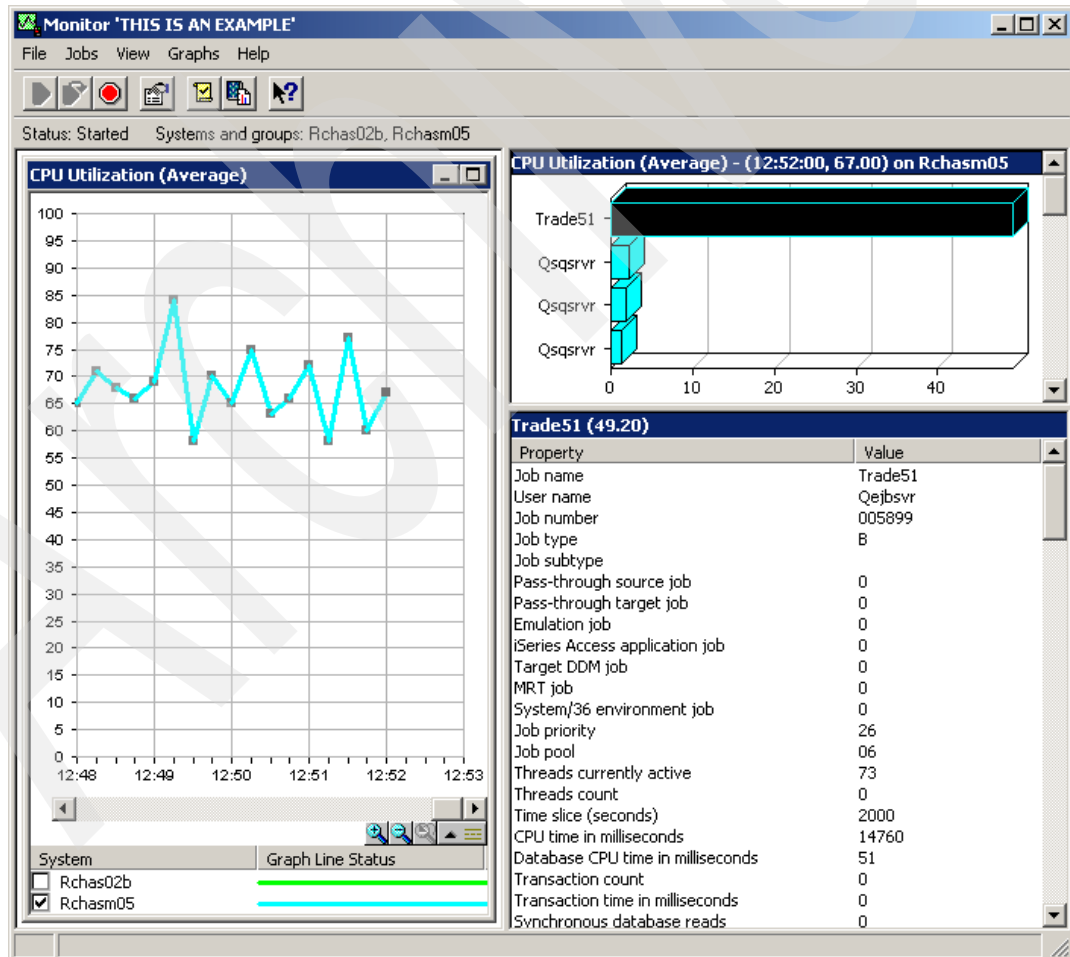


Figure 4-11 Sample Management Central view

Creating a new monitor with iSeries Navigator

Follow these steps to create a new monitor:

1. Launch the iSeries Navigator.
2. Expand your Management Central.
3. Expand **Monitors**.
4. Select **System**, right-click, and select **New Monitor** as shown in Figure 4-12.

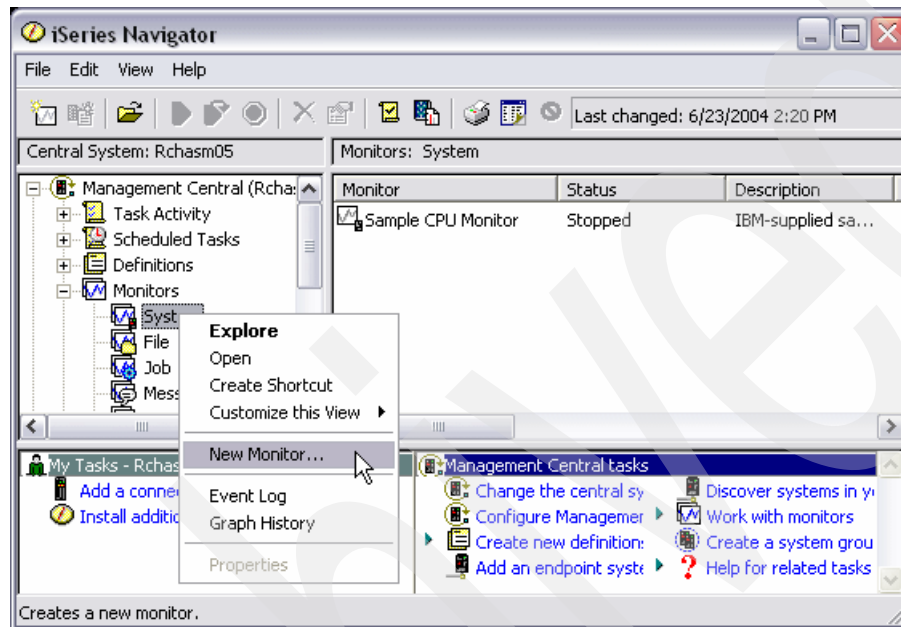


Figure 4-12 Creating a new system monitor

5. The New Monitor opens. Complete these steps:
 - a. Click the **General** tab, and enter a name for the monitor.
 - b. Click the **Metrics** tab. Select the metrics you want to observe and specify (see Figure 4-13):
 - Collection interval of:
 - 15 seconds
 - 30 seconds
 - 1 minute or
 - 5 minutes
 - Maximum graphing value
 - Display time

You may choose to include all 25 available metrics to your monitor. However, we recommend that you select only the key metrics that you need to analyze your current situation. Every metric is shown as a separate pane in the window, so the more metrics you specify, the smaller each pane is.

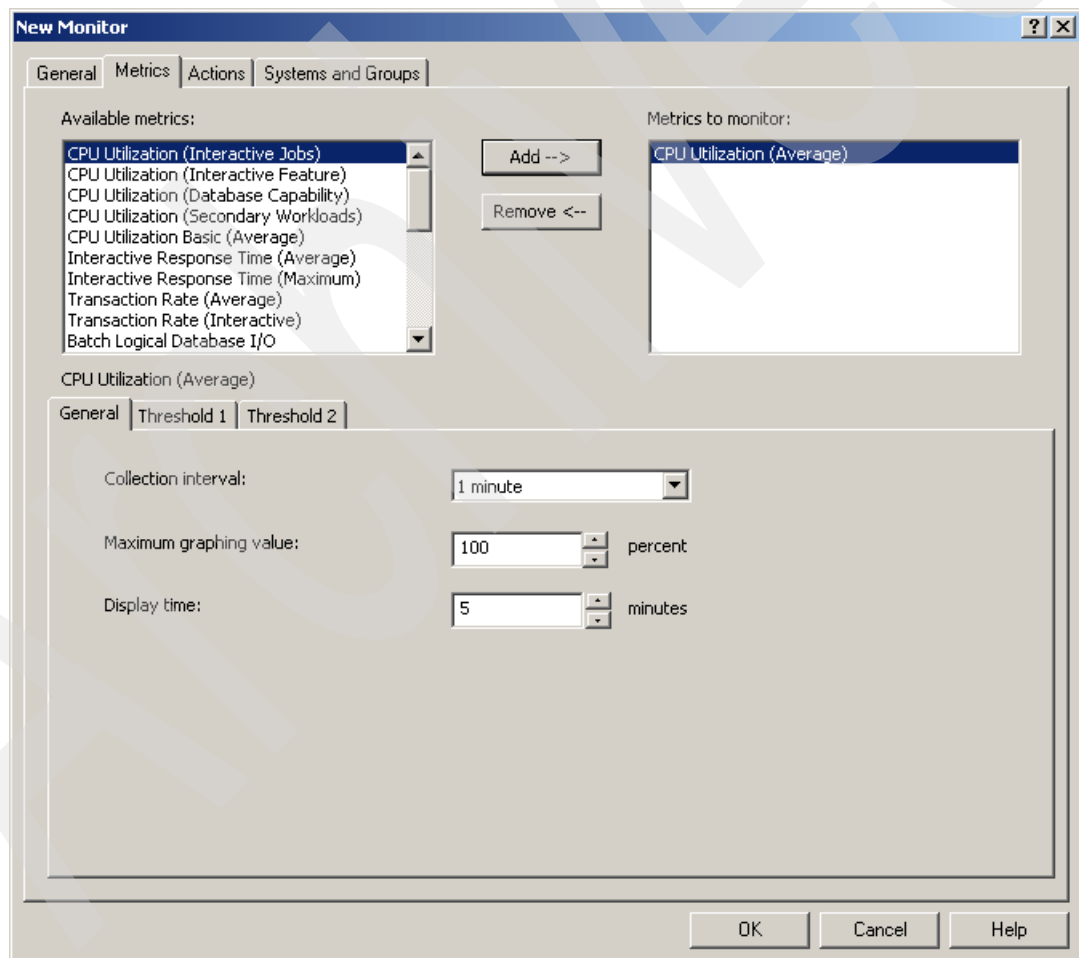


Figure 4-13 Selecting the metrics to observe

- c. Click the **Systems and Groups** tab. Complete these tasks:
 - i. Click the **Browse** button.
 - ii. On the Browse Systems and Groups window (Figure 4-14), select the systems that you want to monitor and click **Add** to add them to your monitor. Then click **OK**.

Tip: Use separate colors for separate systems if you have multiple systems on one monitor. Right-click the central system, select **User Preferences** and set the properties per graph.

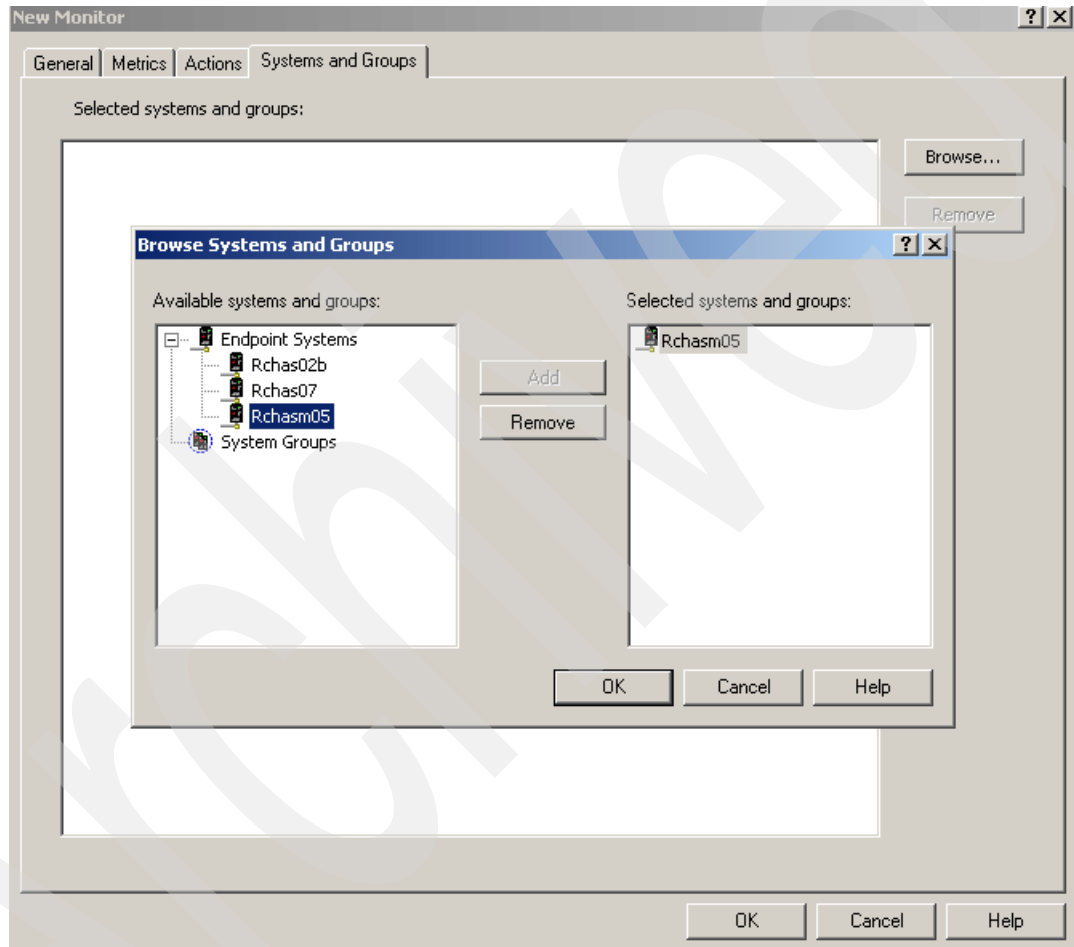


Figure 4-14 Selecting the systems to be monitored

- d. Click **OK**.
6. Double-click your monitor name in the iSeries Navigator.

7. In the System Monitor window (Figure 4-15), click the green Start button to start the monitor.

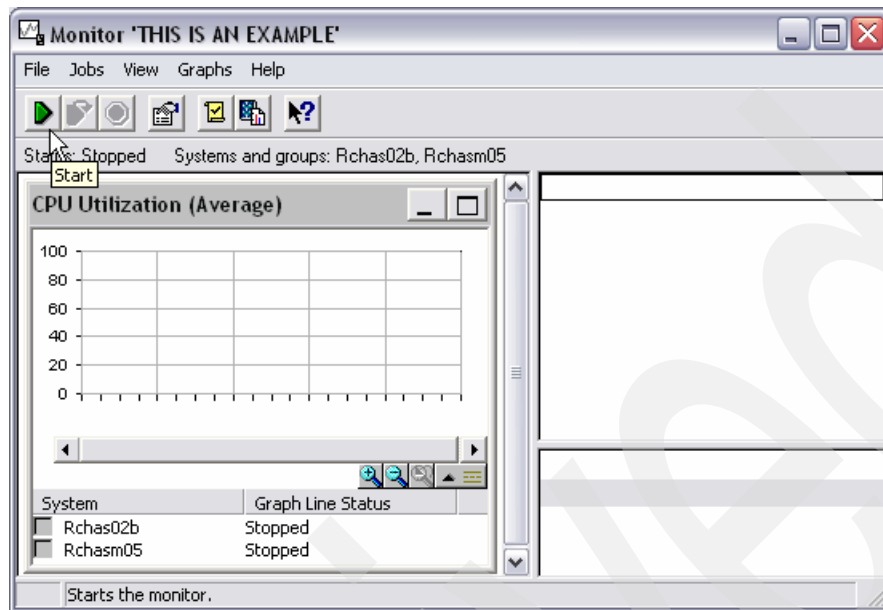


Figure 4-15 Starting the System monitor

4.2.6 iDoctor

The iDoctor suite consists of tools to collect data on a system with a performance problem and tools to analyze that data either on the same system or on a separate system. There are two sets of interfaces to the iDoctor analysis tools:

- ▶ The graphical user interface (GUI)
- ▶ The 5250 display interface

You use the GUI when you must analyze data with a local area network (LAN) connection to the system on which the data resides. You use the 5250 interface when you need to analyze a problem remotely.

iDoctor for iSeries is a suite of tools that consists of the following components:

- ▶ Job Watcher
- ▶ Heap Analysis Tools for Java (also known as Heap Analyzer)
- ▶ Performance Explorer (PEX) Analyzer
- ▶ Performance Trace Data Visualizer (PTDV)

The server-side portion of these components consists of a variety of data collection and analysis programs designed to consolidate performance data in a more usable format. The client-side components for Job Watcher, Heap Analysis Tools for Java and PEX Analyzer consist of GUIs that display the server data in flexible graph and table views. PTDV provides table and tree views of Performance Explorer Collection data with an emphasis on Java analysis.

PTDV and Heap Analysis Tools for Java are offered as a free service on an “as-is” basis. All other components are offered via the “try and buy” approach. You can make a request to purchase the components from the following Web site:

https://www-912.ibm.com/i_dir/idoctor.nsf

Job Watcher

Job Watcher is a service that consists of two parts:

- ▶ Tools for collecting data
- ▶ Tools for analyzing the collected data

The data is collected by a server job QPYSWJOB running in subsystem QIDRJW, stored in database files, and displayed on a client via a GUI.

Job Watcher is similar in sampling function to the system commands WRKACTJOB and WRKSYSACT, where each refresh computes delta information for the ending snapshot interval. In Job Watcher, you can set these refreshes to occur automatically, even as frequently as every 50 milliseconds. Better yet, the Job Watcher harvests the data from the jobs, threads, or tasks being watched in a non-intrusive manner. Job Watcher is available for systems running V5R1 level of OS/400 or later.

Job Watcher is an iDoctor for iSeries tool that returns real-time information about:

- ▶ Selected set of jobs
- ▶ Threads
- ▶ LIC tasks

Job Watcher is particularly useful in the area of problem determination for WebSphere Application Server issues. It may seem that you are often presented with a symptom of a problem, for example:

- ▶ High CPU utilization
- ▶ Heap growth
- ▶ WebSphere Application Server just quits responding

A Job Watcher collection can provide detailed information about every thread running under the WebSphere Application Server server in addition to surfacing general statistics for the Java virtual machine (JVM) in which it is running. The performance impact of running the Job Watcher on a system is minimal.

You can download Job Watcher from:

http://www.ibm.com/eserver/iseries/support/i_dir/idoctor.nsf

When you reach this site, select **downloads** on the left pane and then select the appropriate download option.

Some of the information that Job Watcher shows includes:

- ▶ Standard WRKSYSACT-type information:
 - CPU
 - Direct access storage device (DASD) I/O breakdown
 - DASD space consumption, etc.
- ▶ Real user name in the server jobs
- ▶ Seize time
- ▶ Break down of what types of waits occurred in a job or a thread

Job Watcher also depicts data that is not available anywhere else in real time. For example it presents details on the current wait, such as:

- ▶ Duration of the wait
- ▶ Object being waited for
- ▶ Conflicting job information

The data created by the tool is summarized in many different types of reports and graphs via the GUI client. The client provides a quick picture of what is happening on a per-thread basis when multiple different threads are being watched. You have the flexibility to select a job or an interval and then drill down for the details while the watch is in progress or after the watch ends. You may even save the collected data and restore it to another system and analyze it there.

System-wide Job Watch versus a job-specific Job Watch

There are two types of Job Watch collections: job specific and system wide. *System-wide* collection is a good starting point to use if you do not know which jobs are running poorly. As Job Watch monitors the entire system, it helps you find the poorly performing jobs.

A *job-specific* Job Watch is better if you know that a particular job or set of jobs are running poorly. It gives you more information from this type of Job Watch, such as the call stack for each interval for each job or thread that is being watched.

The Job Watch wizard provides the interface for defining and starting Job Watch collections. The wizard gives you the option to define a trigger (or reuse or modify an existing trigger) definition. The trigger determines the conditions for which the Job Watch will call a program or begin collecting data. For example, you can use a trigger to call a program that will page a technician when the fault rate for a production job becomes high.

When selecting the jobs to watch within the wizard, a window of job statistics across the entire system is shown. These statistics are detailed and useful for determining for which job or jobs to start a more detailed Job Watch.

You may even start a Job Watch over a job or jobs on a job queue if desired. No data is collected until the job runs.

The wizard provides optional data collection options for Structured Query Language (SQL) statements, socket information, activation groups information, and call stack details. You may optionally turn off or on each of these for inclusion in the Job Watch.

Starting a Job Watcher collection

To start a Job Watch collection, follow these steps:

1. From the Microsoft Windows® desktop, select **Start → Programs → iDoctor for iSeries → Job Watcher** to start a Job Watcher collection. If you use the Job Watcher frequently, consider creating a copy of the icon to the Quick Launch toolbar.
2. Right-click to select the system you want work as shown in Figure 4-16.

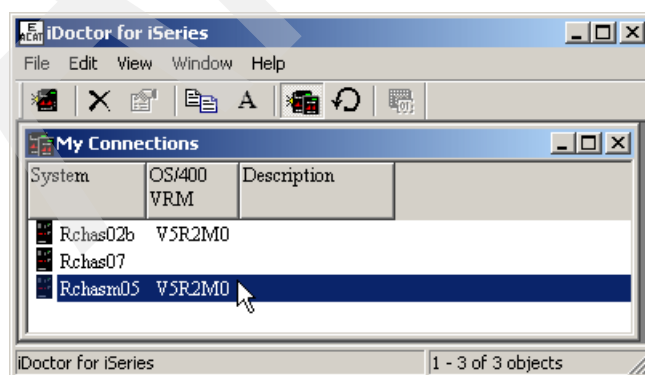


Figure 4-16 Starting the iDoctor

3. Because iDoctor uses iSeries security, you see a signon window as shown in Figure 4-17. Type your user ID and password and then click **OK**.

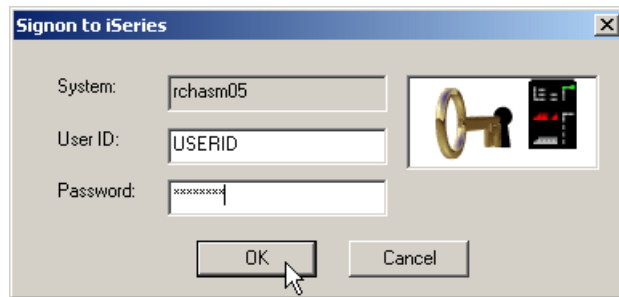


Figure 4-17 Signing on to the system

4. The iDoctor Components window (Figure 4-18) opens. Either highlight a component and click the Launch button or double-click a component to start it.

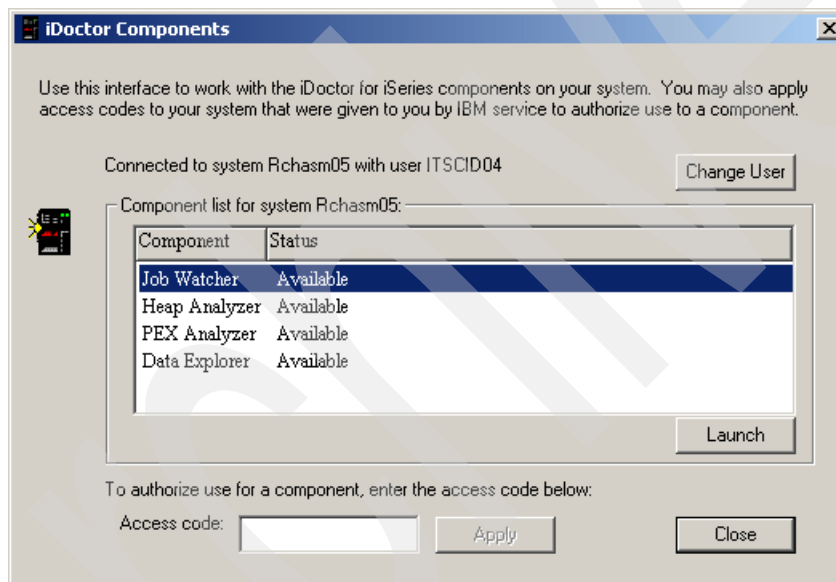


Figure 4-18 Selecting the iDoctor component to work with

- After you launch Job Watcher, you see a panel that shows the libraries that have previous Job Watcher collections. Select a library by either expanding the navigational tree in the left panel or by double-clicking a library (shown as a folder) in the right panel. Figure 4-19 shows an example of navigating through the left panel and right-clicking.

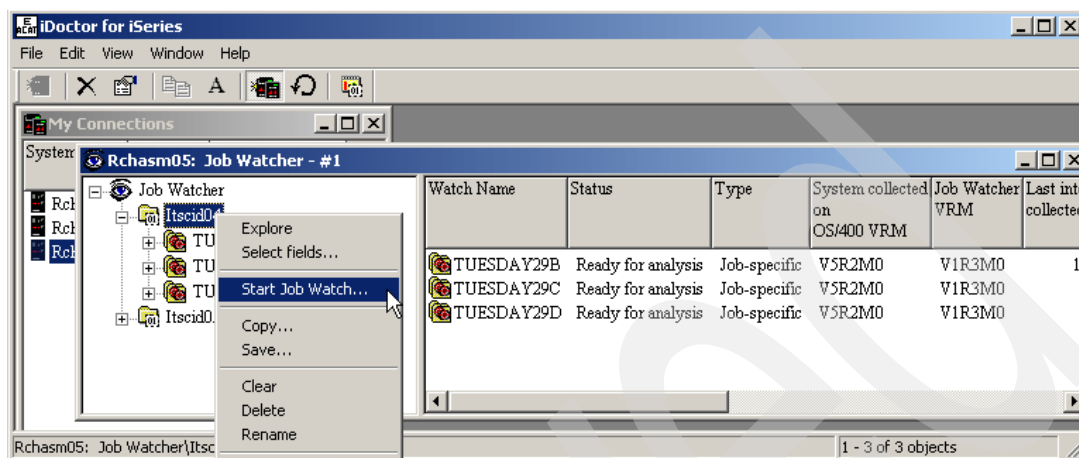


Figure 4-19 Starting a new Job Watcher

Starting a Job Watch

Job Watches are created using the Job Watch wizard within Job Watcher. They contain summarized and detailed views of many different types of data over the jobs that are being watched. Figure 4-20 shows the Welcome panel of the Job Watch wizard. This example illustrates how to create a Job Watch for a specific job, TRADE51.

- On the Job Watch Wizard Welcome panel, select the type of Job Watch that you want to create. Click **Next**.

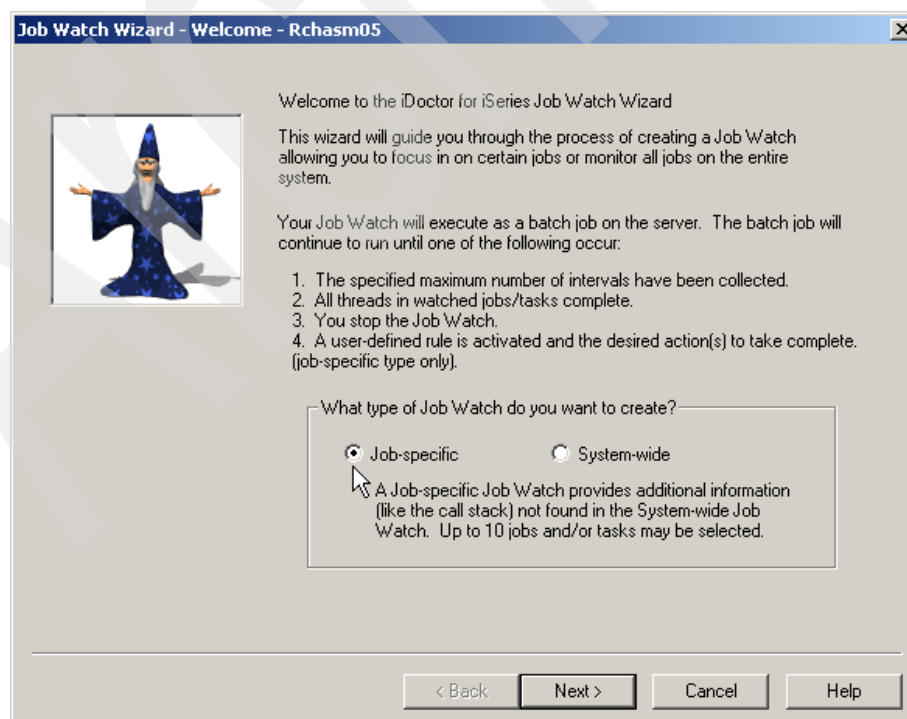


Figure 4-20 Selecting between starting a job specific or system wide Job Watcher

2. On the Job Watch Wizard Options panel (Figure 4-21), provide this information:
 - a. Give a name for the Job Watch.
 - b. Specify the collection interval.
 - c. Specify the number of collection intervals.
 - d. You may also select the option **Collect as fast as possible**. It is a good option to start with since it increases the number of collection snapshots and maximizes the chances of collecting the data required for a deeper analysis.

Attention: Setting the Collect as fast as possible option on can cause a serious performance impact on the system while collecting the data. This is based on the nature of the data collection - the tool adds data to collection members as often as it takes snapshots of the system activity.

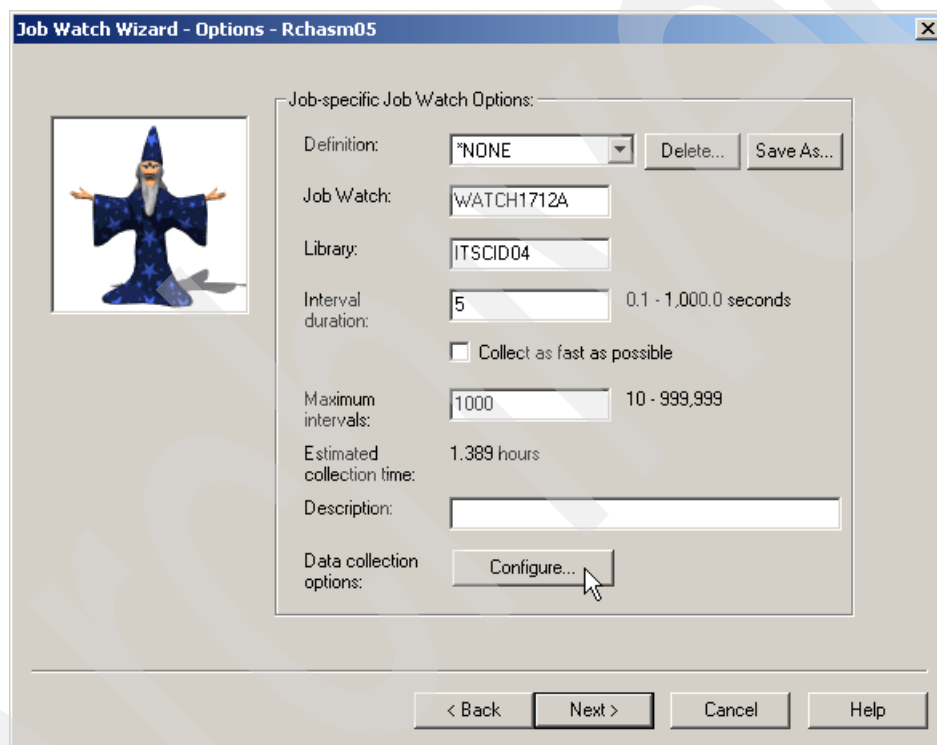


Figure 4-21 Specifying the Job Watcher name and collection options

3. You click the **Configure** button
4. The Data Collection Options window (Figure 4-22) opens.
 - a. You may specify between the following levels of data to collect for SQL statement data:
 - None
 - SQL statements only
 - SQL statements and host variables
 - SQL statements, host variables, prepared statements, and open cursors
 - b. For communication data, you may specify the following levels:
 - None
 - Communication socket and IP information
 - Communications detail and related job information

c. When collecting the data for activation group details, you may specify:

- None
- Capture once at startup
- Collect every interval
- Collect every n-th interval

d. When collecting for the Call stack details, you may specify:

- None
- Collect every interval
- Collect every n-th interval

Attention: The more detailed data you collect, the more disk space you will use. This may become an issue if the system has only a small amount of disk space available.

e. Click **OK**.

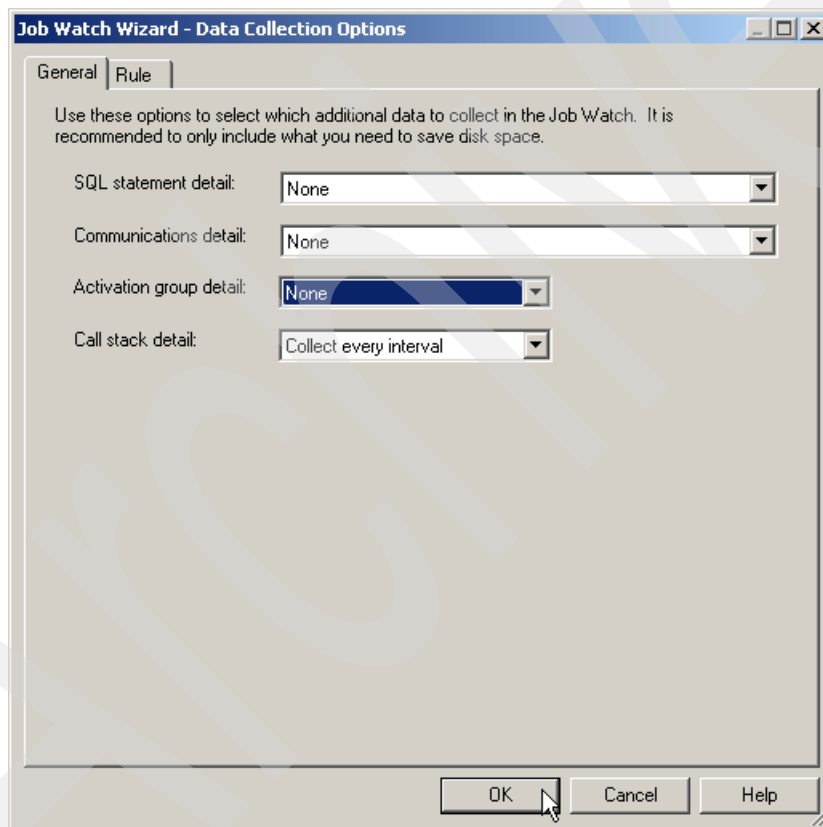


Figure 4-22 Specifying the type of data to collect

f. Back on the Options panel (Figure 4-21), click **Next**.

5. The Job/Task Option panel (Figure 4-23) opens. On this panel, make a job selection. In this example, we select the **Selected jobs** radio button. Click **Next**.

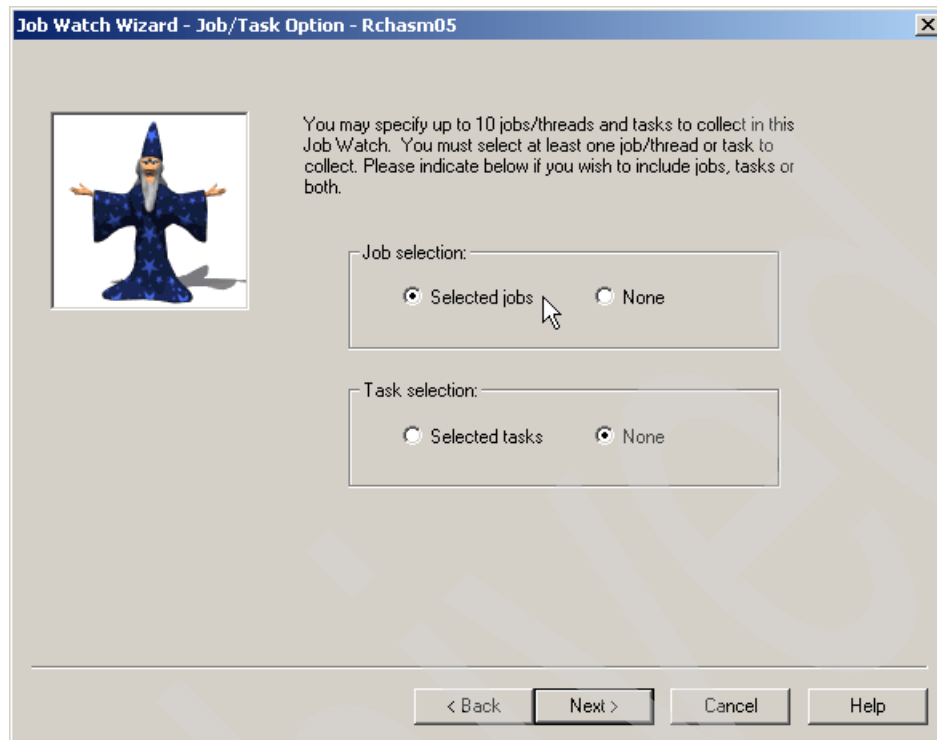


Figure 4-23 Creating the Job Watcher collection for specific jobs

6. The Add Jobs window (Figure 4-24) opens. You may either type a full job name or use a wild card (an asterisk) at the end of the job name to search for a job. Click the **Refresh List** button.

Job Watch Wizard - Add Jobs - Rchasm05

Please indicate the jobs you wish to add to your Job Watch:

Job Information:

Name: Status:

User: Current user:

To build the list of jobs to select from, type a generic job name and job user name and click 'Refresh list'

Figure 4-24 Specifying the jobs to include to the Job Watcher collection

The window changes to display the view shown in Figure 4-25. Click the **Add Selected** button. Then click **Close**.

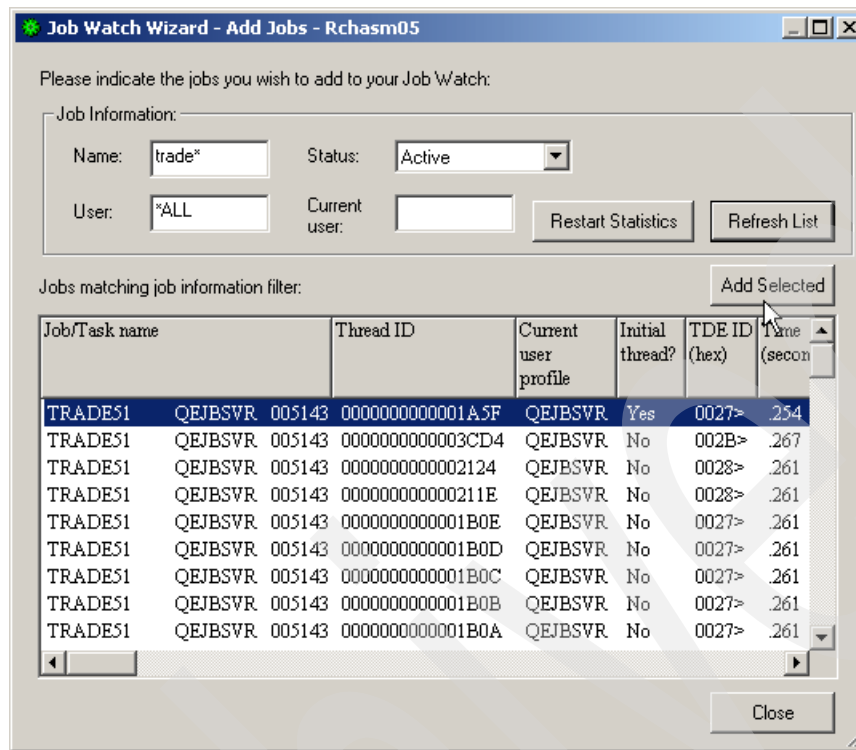


Figure 4-25 Adding specific jobs to the Job Watcher collection

7. The Job Selection panel (Figure 4-26) opens. Select all the threads and click **Next**.

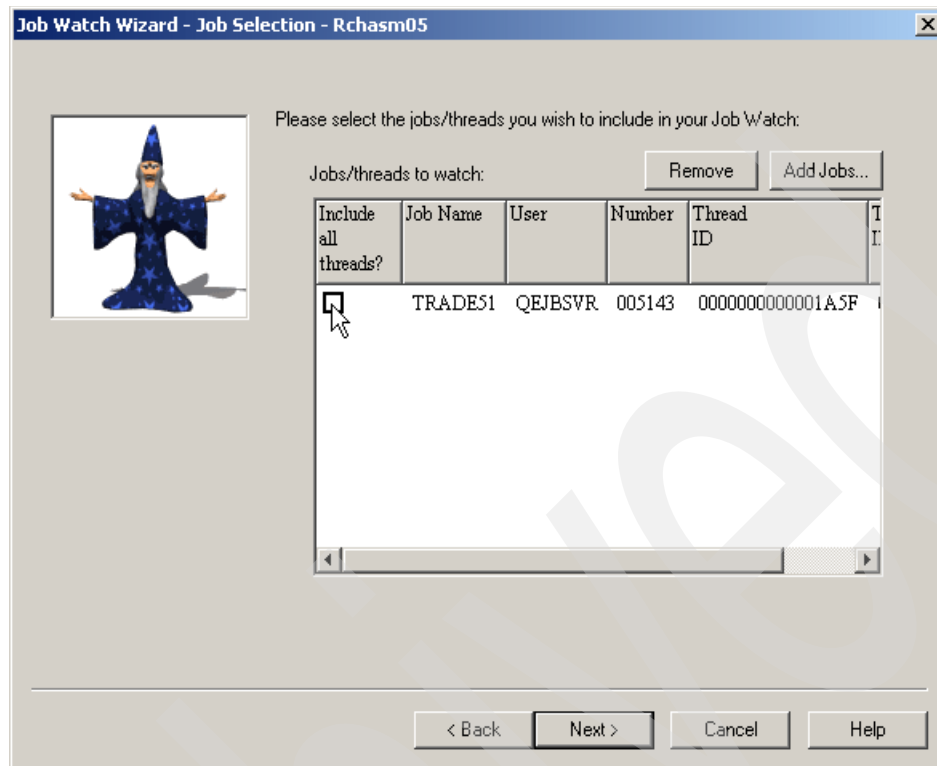


Figure 4-26 Including all the threads to the Job Watcher

- Now you see a summary panel (Figure 4-27) that shows all the selections that were made in the previous steps. If you are satisfied with the selections, click the **Finish** button to submit the collection job.

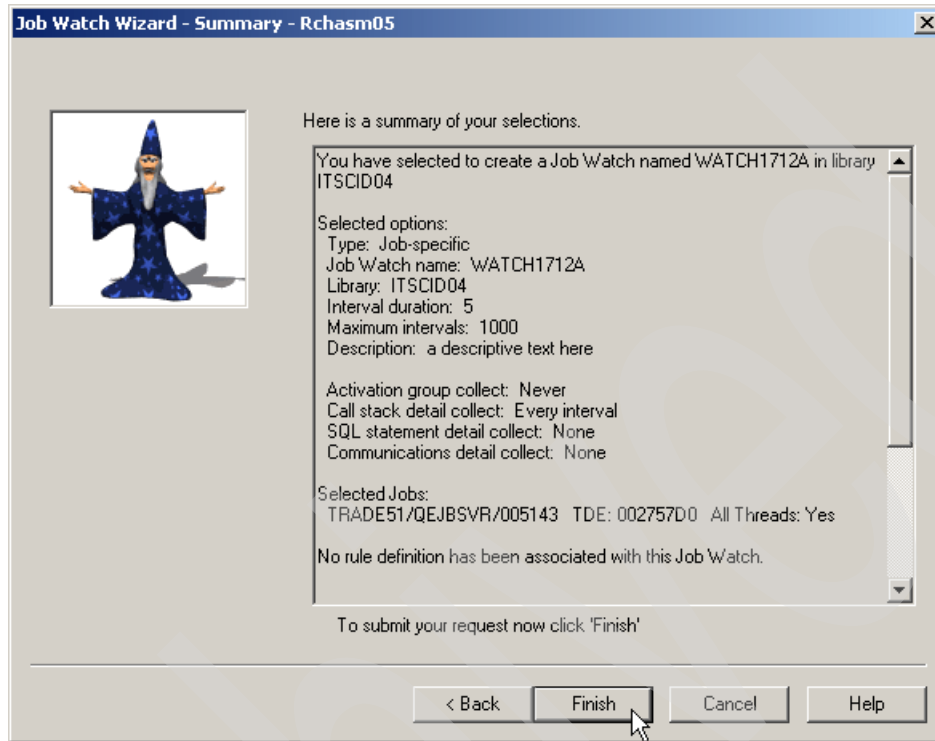


Figure 4-27 Job Watcher Wizard Summary panel

Figure 4-28 shows an example of the available collections in a library.

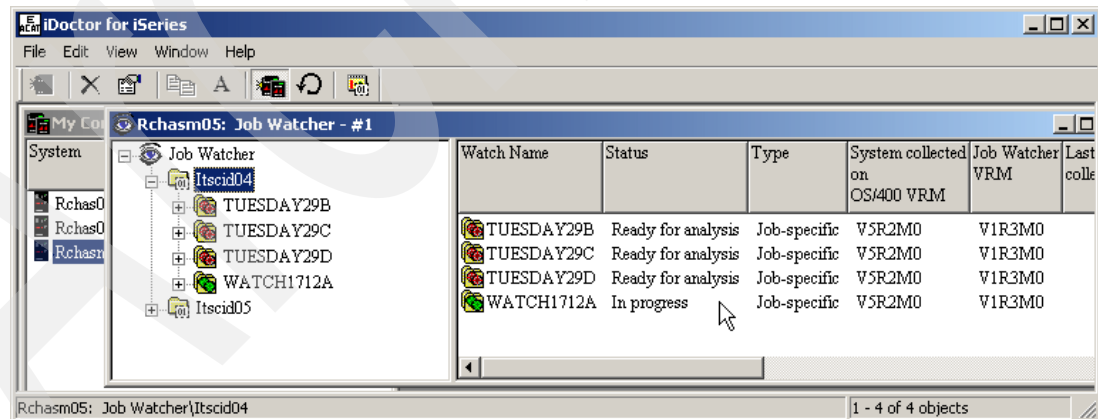


Figure 4-28 Viewing the Job Watcher collections within a library

Working with the Job Watch collections

A Job Watch collection contains several folders, each reporting information about the collection in different way. All of these folders are available (with the exception of the Current Activity folder) regardless of whether the collection is active. This gives you the option to either analyze your collection in real time or to save the data and analyze it after the collection has completed.

The *Watched Threads* folder shows a list of threads or tasks that is being watched. This list indicates whether the thread is still active. You can right-click any thread in this list to see a menu option with a variety of different graphs and tables that show intervalized statistics about the particular thread.

The *Current Activity* folder offers several different ways to look at the most recent interval of data per thread being watched. With each refresh of this view, you can see what's happening in real time.

The *Summarized Reports* folder displays different views, each summarizing what happened in each thread over the last *n* intervals, where *n* is a configurable number. For example, this feature lets you see what your jobs did in the last 10 seconds or in the last 5 minutes. You may right-click any of the jobs in these reports to drill down on them and run one of the detail graphs or reports. This shows each interval of data for a particular thread.

The *Summarized Graphs* folder displays different stacked bar graphs, with each bar representing one of the threads in the Job Watch. These graphs show the same types of data as those shown via the Summarized Reports folder, but in a bar graph format.

The *Run/wait signature graph* shows precisely how much time a job or thread was using CPU versus experiencing different types of waits. Each color or section of the bar represents a different type of wait.

While working with any Summarized graph, you can right-click one of the bars to see a detailed or intervalized report. The detail report shows the same types of information but on a per interval basis instead of over the last *n* intervals. When working with any intervalized report or graph, you can drill down further into any particular interval to see the call stack. Or you can see additional details about the interval such as the amount of time spent in each type of wait or the object the thread was waiting on (if applicable).

When displaying the list of Job Watches, you may reorder or configure the fields shown in any manner you choose. You may also delete a Job Watch collection if it is completed or stopped if it is active. Plus you can right-click a collection to display either reports or graphs as shown in Figure 4-29.

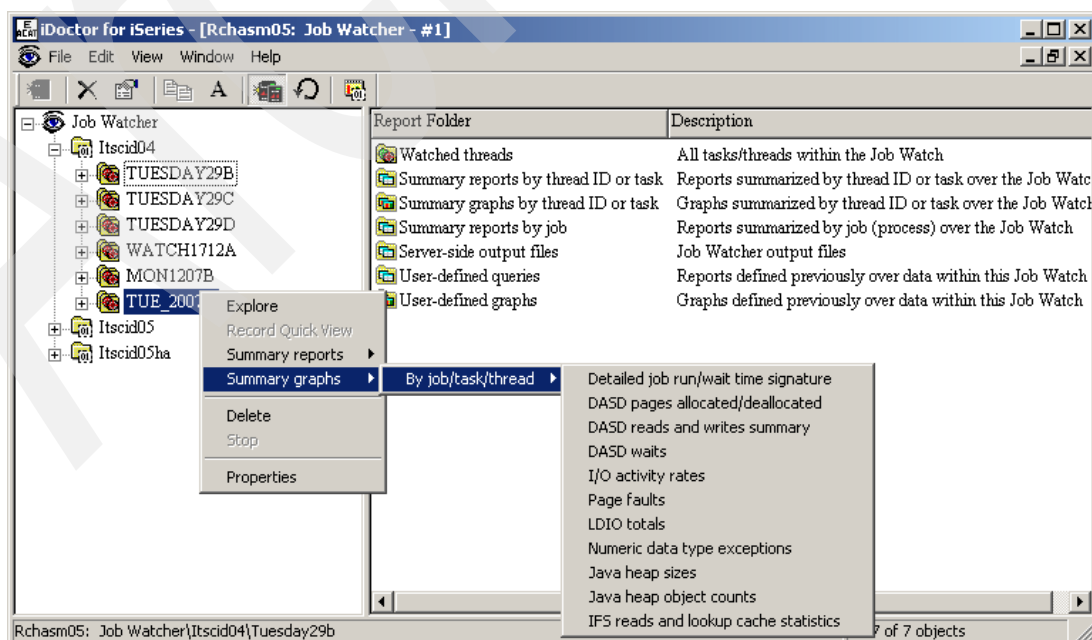


Figure 4-29 Summary graphs available per a thread

Viewing the Job Watcher graphs

You can configure Job Watcher graphs to automatically refresh every n seconds. You can use this as a visual monitor for your job or jobs.

Graphs offer several drill-down type features. You can click a bar or right-click and use the menu options that appear to see additional information about the particular job or interval within a job. Simply place your cursor over any bar or color to see its value.

Tip: Right-click a graph for a menu. Select the graph properties and then select **General/Y-axis** to set the graph type.

Viewing Job Watcher reports

Like Job Watcher graphs, you can configure Job Watcher tables to refresh automatically every n seconds. You may also hide and reorder fields in the file to any sequence that you choose using the drag and drop functionality.

You may export the current report data or the current selection to the PC in the following formats:

- ▶ Comma separated
- ▶ Tab-delimited
- ▶ Rich text

Job Watcher also supports viewing files that contain millions of records. Only the data that needs to be read into the PC for display is transferred at a time. As you scroll through the file, a page of data is read in one block at a time, which provides fast response times. If you jump to the end of the file, only the last page is read instead of the entire file. You may easily copy any selected data to the clipboard.

There is an option to vertically view any record of data, or to vertically view and compare multiple records. This allows you to see more of the fields for a single record at one time if there are many (dozens) of fields in the file.

You have the ability to perform a positional search for a specific text string within a file or query. Plus you can customize the fonts used in the reports.

Call stacks

Accessing the call stack for a thread in the Job Watcher is done in a non-intrusive manner. It does not require holding and releasing the thread.

Job Watcher can collect up to 50 levels deep of call stacks. This includes long procedure names which can be invaluable in telling programmers where to look to solve problems.

You can search the call stack data by using the Find button in the Call Stack window. This is useful for finding where a specific procedure, program, or module name was called.

You also can have multiple call stack windows open at one time to compare different stacks at the same time. Figure 4-30 shows the graphical interface into the files into which the collection data is written.

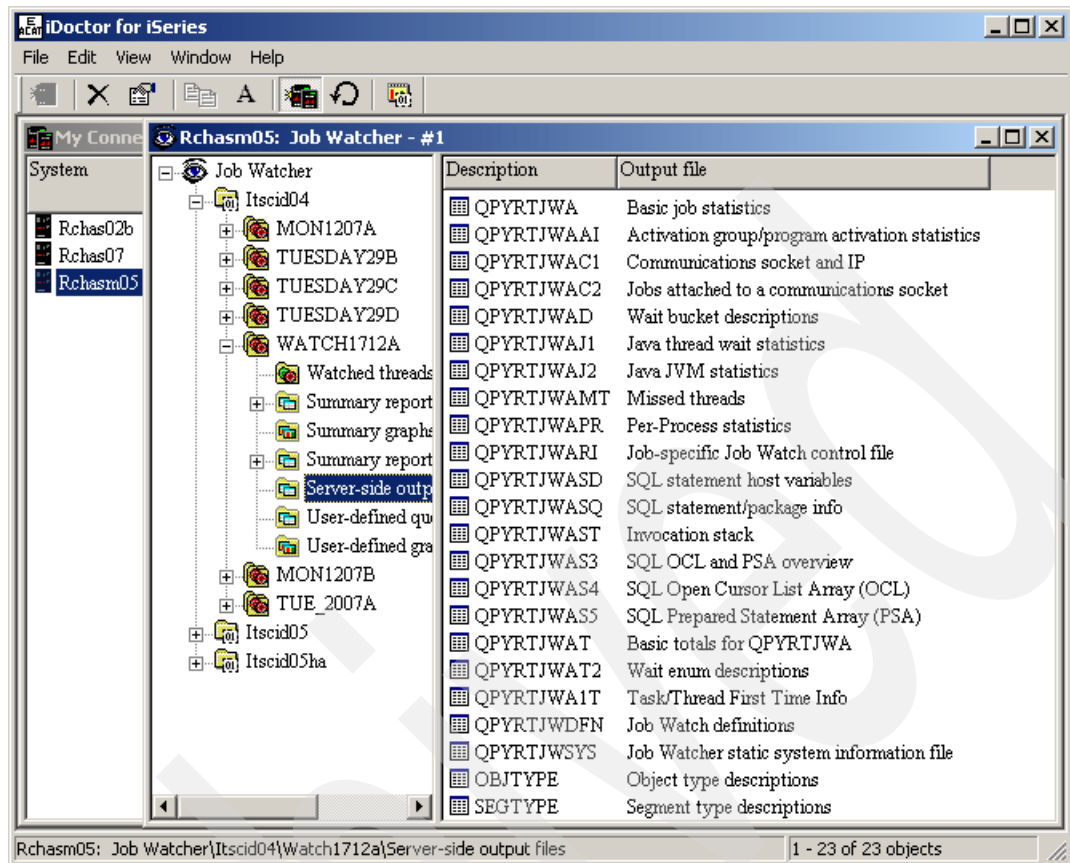


Figure 4-30 Graphical interface to the iDoctor output files

Using the Job Watcher via 5250 interface

To start the Job Watcher collection via a 5250 display in V5R2, you enter the following command:

```
ADDLIB QPYRTJW
SBMJOB CMD(WCHJOB ACTION(*FIRSTTIME) MBR(JWDATA) LIB(JWTEST) JOBSTASKS((JVMJOB 999999))
AUTO(*FULLSPEED) AUTOACTGRP(*NEVER) AUTOLIMIT(5000))
```

When using the 5250 interface to analyze the data, you must use either queries or SQL. Table 4-1 provides information about the files to investigate.

Table 4-1 Job watcher files in R520 and R530

| File description | R520 QPYRTJW | R530 QIDRWCH |
|---|--------------|--|
| Object type descriptions | OBJTYPE | QAIDROT |
| Basic statistics per a thread or an LIC task | QPYRTJWA | QAPYJWTDE |
| Task or thread first time information | QPYRTJWA1T | |
| Activation group or program activation statistics | QPYRTJWAAI | QAPYJWAIPA, QAPYJWAIGP, QAPYJWAIHP |
| Communications socket and IP | QPYRTJWAC1 | QAPYJWSKTC |
| Jobs attached to communications socket | QPYRTJWAC2 | QAPYJWSKJB |

| File description | R520 QPYRTJW | R530 QIDRWCH |
|--|--------------|--------------------------|
| Wait bucket descriptions | QPYRTJWAD | QAPYJWBKT |
| Java thread wait statistics | QPYRTJWAJ1 | QAPYJWJVTH |
| Java JVM statistics | QPYRTJWAJ2 | QAPYPJWJVM |
| Missed threads | QPYRTJWAMT | |
| Statistics per a job | QPYRTJWAPR | QAPYJWPRC |
| Job-specific Job Watch control file | QPYRTJWARI | QAIDRJWRI |
| SQL open cursor list and prepared statement array overview | QPYRTJWAS3 | |
| SQL OCL array | QPYRTJWAS4 | QAPYJWSQLO |
| SQL PSA | QPYRTJWAS5 | QAPYJWSQLP |
| SQL statement host variables | QPYRTJWASD | QAPYJWSQLH |
| SQL statement and package information | QPYRTJWASQ | QAPYJWSQL |
| Invocation stack | QPYRTJWAST | QAPYJWSTK, QAPYJWPROC |
| Wait enum descriptions | QPYRTJWAT2 | QAIDRJWENM |
| Rule definitions | QPYRTJWRDB | Any TXT file |
| Static system information file | QPYRTJWSYS | |
| Segment type descriptions, target user | SEGTYPE | QAIDRST |
| Basic interval information | | QAPYJWINTI |
| Java Task Dispatch Element (TDE) data | | QAPYJWJTH |
| Basic collection and system | | QAPYJWRUNI |
| TDE status for all TDEs and intervals | | QAPYJWSTS |

Heap Analysis Tools for Java (Heap Analyzer)

Usually, the first symptom of a heap growth-related problem is a gradual or sudden increased rate of paging in the pool where the Java jobs are running. This is most often caused by uncontrollable growth of the heap size. If you keep adding memory to the pool, you treat the symptoms, but do not remove the problem itself.

Heap Analysis Tools for Java provides tools for doing the heap analysis, advanced debug, and application review. You use this tool in two stages:

1. Collect the data.
2. Analyze the data.

The following sections explain the steps for collecting and analyzing data. In these steps, you must choose from three different collection options:

- Object table dump
- Object create profile
- Object root finder

The Heap Analysis Tools component of the iDoctor is used to perform Java application heap analysis and object create profiling (size and identification) over time. Heap Analysis Tools for Java includes information about:

- ▶ JVM heap growth or size
- ▶ The objects being created (type of object, count and object size, object heap size)
- ▶ The application “Heap Footprint®” for memory sizing and performance considerations

Heap Analysis Tools for Java includes a call stack for every snapshot when run in profile mode so the created objects can be correlated to the application functions.

Object table dump

If you are working on a problem, it is best to collect at least one object table dump before the problem surfaces with the JVM and becomes active. This helps to establish a base line to compare the following dumps. When you see that the heap is showing growth, you want several object table dumps to use this to:

- ▶ Establish a pattern of growth
- ▶ Identify the object class type and the size of the object you feel may be causing the growth

The frequency of the dumps depends on how fast the leak is. For example, if you have heap growth at 200 MB per hour, then a dump every two hours may be the right choice. If your growth is more like 1 GB per hour, then you should have a dump at least every 45 minutes. These dumps are relatively small in size, so space should not be an issue. The main concern is that you want to obtain the data prior to any heavy paging starts.

Collecting data for object table dump

Starting the Heap Analysis Tools for Java is almost similar to starting Job Watcher (see “Job Watcher” on page 41). After selecting the Heap Analysis Tools for Java component, you may start working with an existing collection as shown in Figure 4-31.

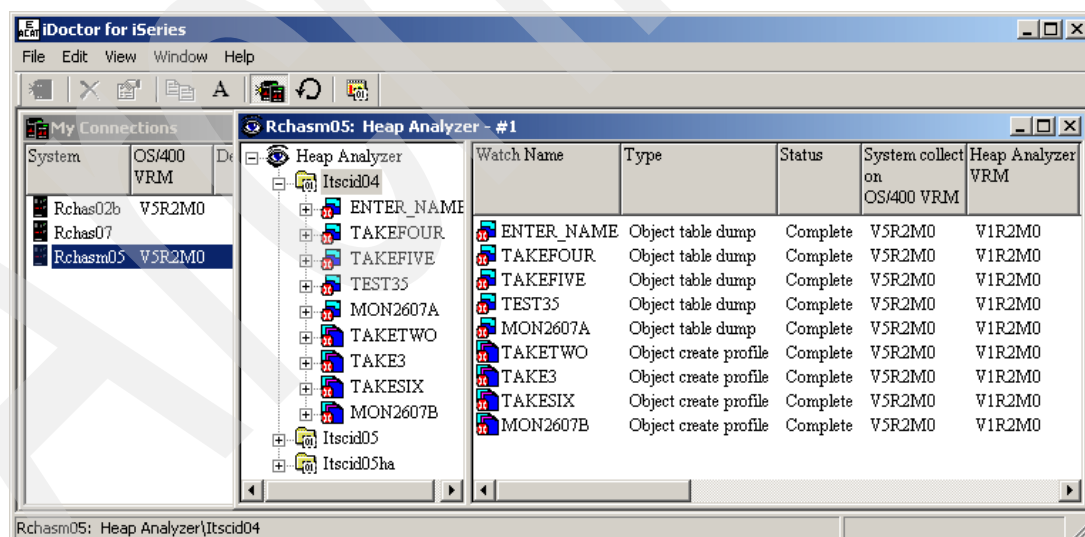


Figure 4-31 Selecting a Heap Watcher Collection to work with

You may also start a new Heap Watcher collection as shown in Figure 4-32.

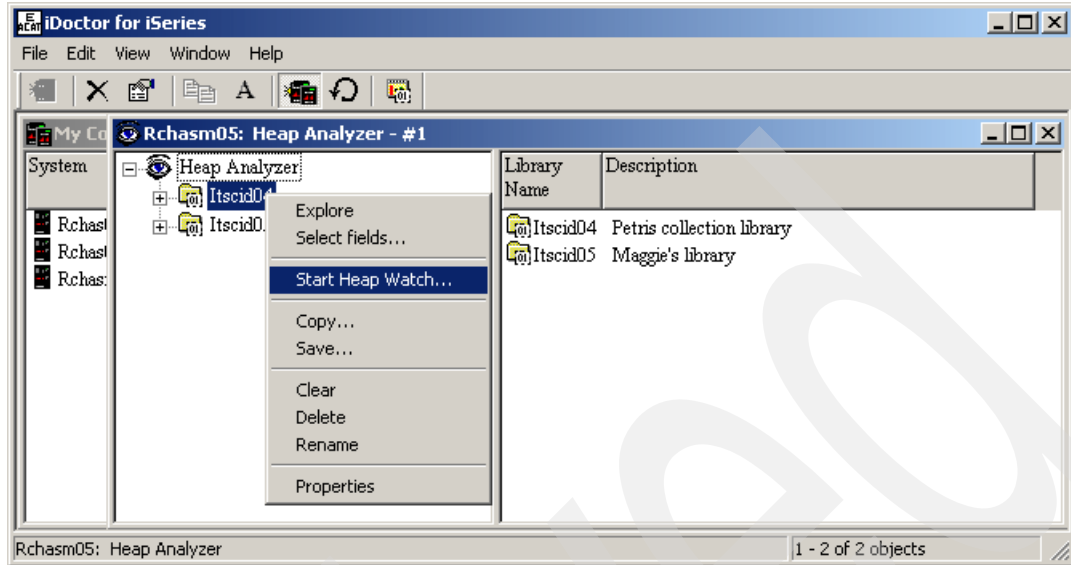


Figure 4-32 Starting the Heap Watcher

1. When you start Heap Watcher, the Heap Watch Wizard window opens showing the Welcome panel (Figure 4-33). Based on the selections done on this panel, you start different collections at later stages. In this example, we select **Object table dump**. Click **Next**.

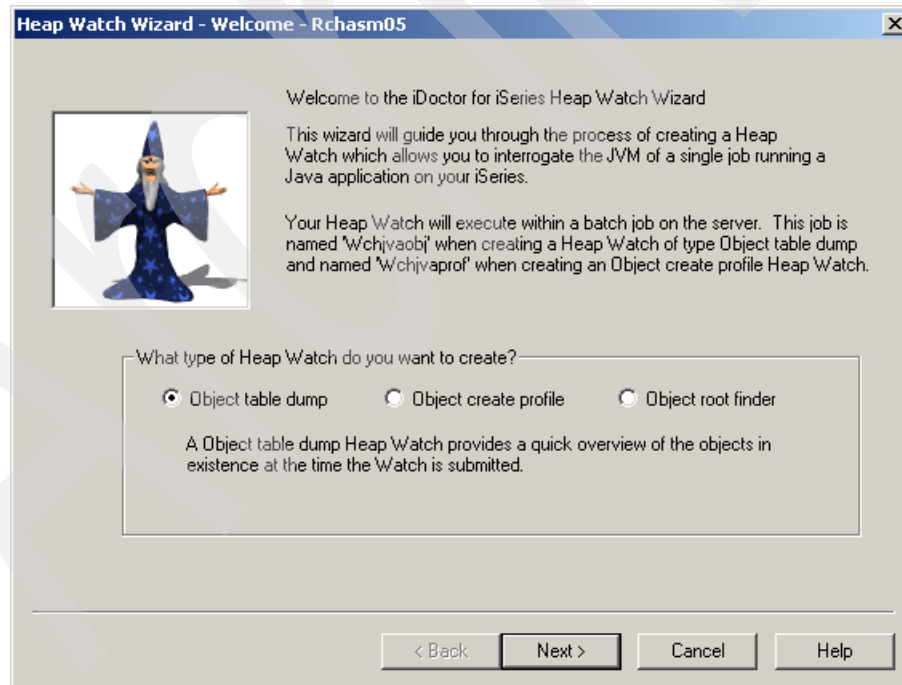
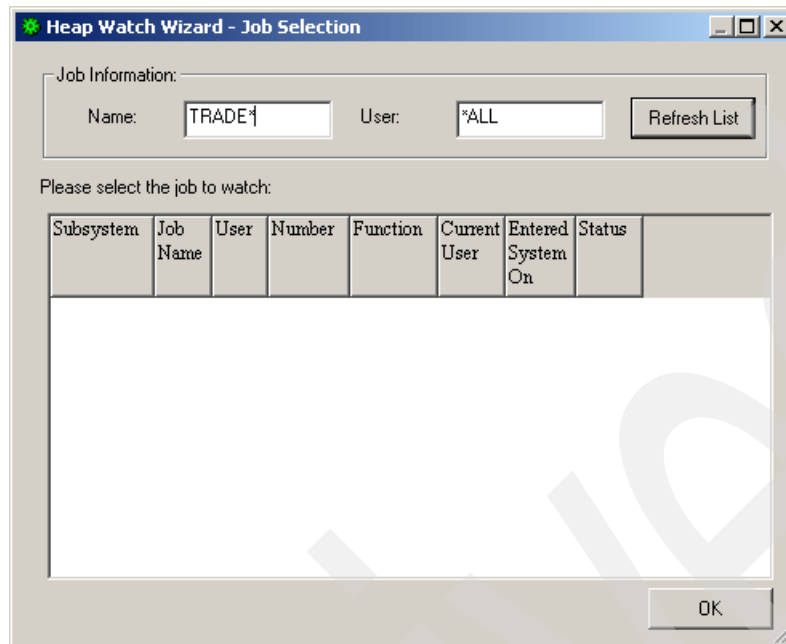


Figure 4-33 Heap Watch Wizard: Welcome panel

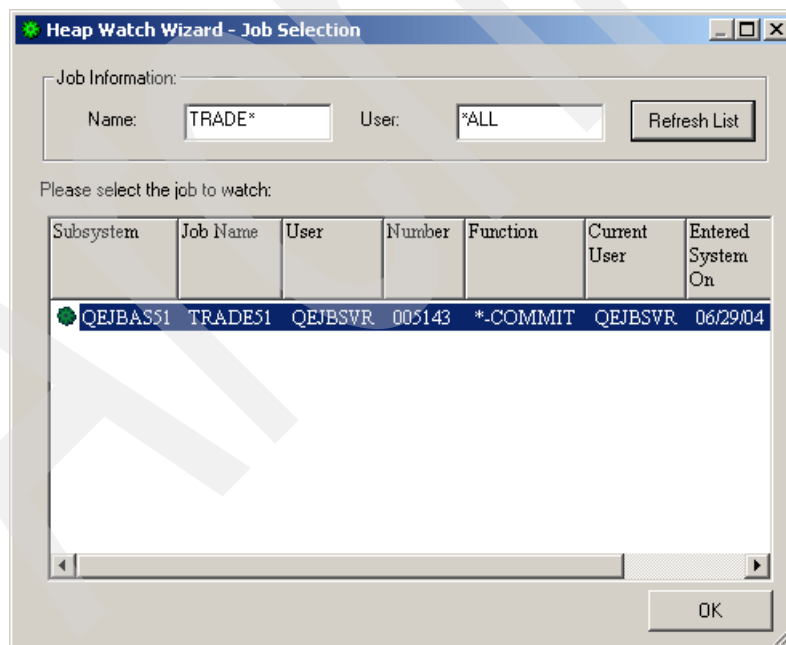
2. The Job Selection window (Figure 4-34) opens. Enter a partial job name and either press the Enter key or click the **Refresh List** button.



The screenshot shows the 'Heap Watch Wizard - Job Selection' window. It has a title bar with a green icon and standard window controls. Below the title bar is a 'Job Information:' section with two text boxes: 'Name:' containing 'TRADE' and 'User:' containing '*ALL'. To the right of these is a 'Refresh List' button. Below this section is the instruction 'Please select the job to watch:' followed by a table with the following headers: Subsystem, Job Name, User, Number, Function, Current User, Entered System On, and Status. The table body is currently empty. At the bottom right of the window is an 'OK' button.

Figure 4-34 Selecting a job to monitor

3. Now you see a list of jobs similar to the one shown in Figure 4-35. Select the job and click **OK**.



The screenshot shows the same 'Heap Watch Wizard - Job Selection' window, but now the table contains one row of data. The 'Name' field is updated to 'TRADE*'. The table row is highlighted with a blue background and contains the following values: Subsystem (QEJBAS51), Job Name (TRADE51), User (QEJB5VR), Number (005143), Function (*-COMMIT), Current User (QEJB5VR), and Entered System On (06/29/04). The 'Status' column is empty. The 'OK' button remains at the bottom right.

| Subsystem | Job Name | User | Number | Function | Current User | Entered System On | Status |
|-----------|----------|---------|--------|----------|--------------|-------------------|--------|
| QEJBAS51 | TRADE51 | QEJB5VR | 005143 | *-COMMIT | QEJB5VR | 06/29/04 | |

Figure 4-35 Adding a job to the collection

4. Verify the options as shown in Figure 4-36 and click **Next** to continue.

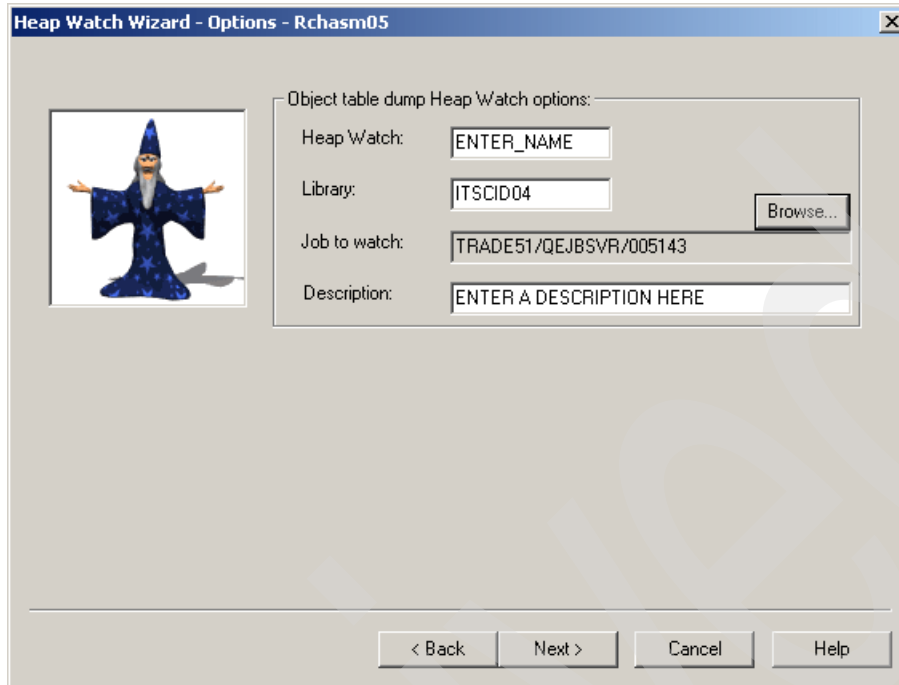


Figure 4-36 Verifying the job to watch

5. You may specify the execution limit (time) as shown in Figure 4-37 and click **Next**.

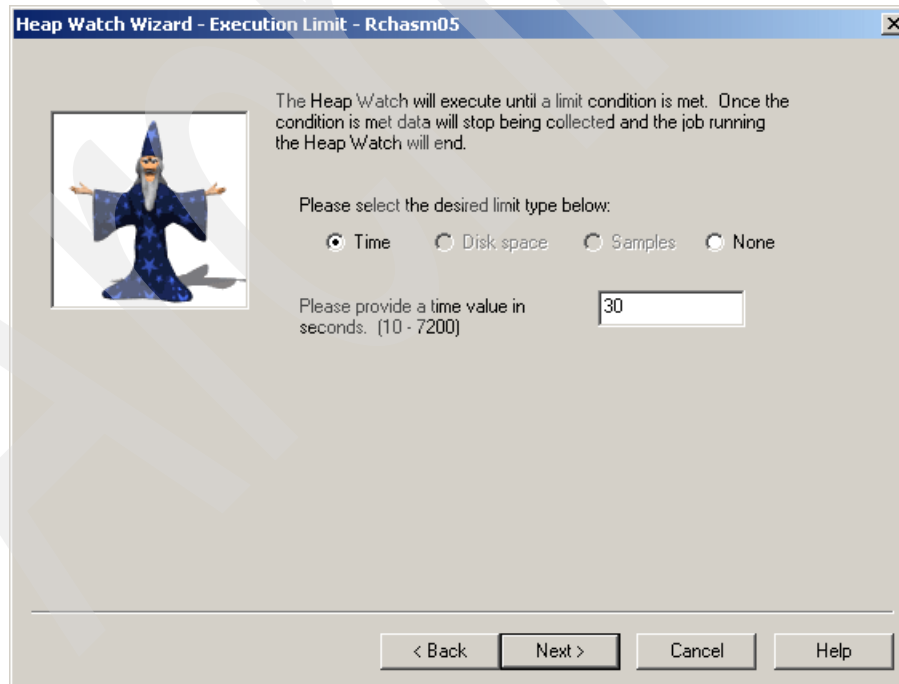


Figure 4-37 Specifying the ending condition for the collection

6. The Summary panel of the Heap Watch Wizard opens as shown in Figure 4-38. Click the **Finish** button. Then the remote command string shown in Figure 4-38 is sent to the previously specified system for execution.

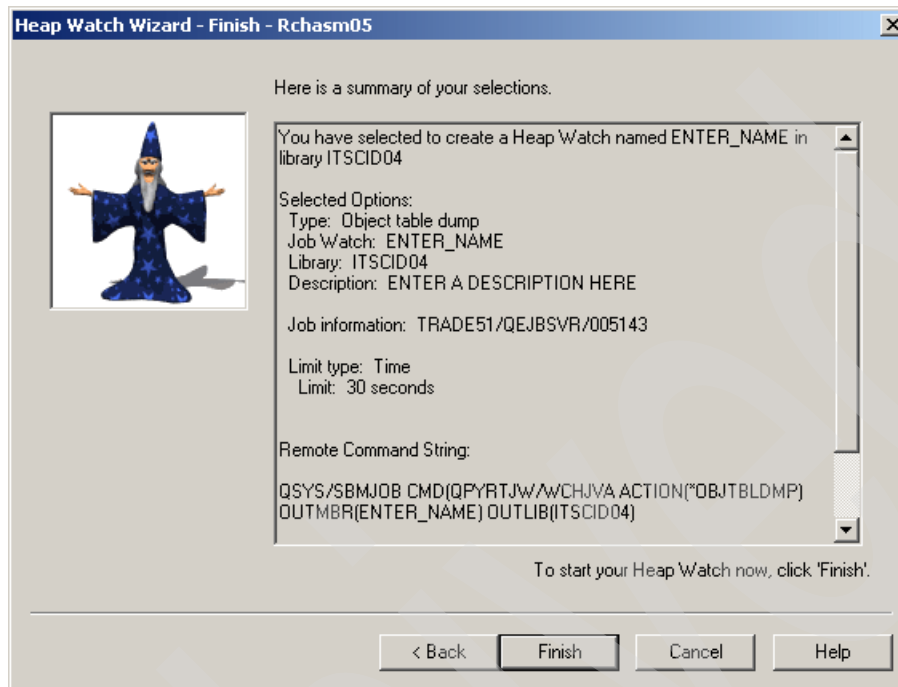


Figure 4-38 Selection Summary panel for object table dump

Collecting data for object create profile

Another good way to understand heap issues is to turn on the object create profile during a period when you see growth. If you can identify an object class that you are interested in from the object table dump or dumps, you can usually use the profiler output to obtain an idea about where the object is being created in the application. You can use the following two commands to start a profiler collection from a 5250 display:

```
ADDLIB QPYRTJW
SBMJOB CMD(WCHJVA ACTION(*OBJCRTPROFILE) OUTMBR(JWDATA) OUTLIB(PROFILE)
JOB(999999/USERID/JVMJOB) TIME(1800)) JOB(*JOB))
```

The graphical interface of doing this is shown Figure 4-39 through Figure 4-42. When you specify to create a new Heap Watch, you follow the wizard shown in Figure 4-39.

1. In the Heap Watch Wizard Welcome panel (Figure 4-39), select the **Object create profile** radio button and click **Next**.



Figure 4-39 Starting collection for object create profile

2. In the Heap Watch Wizard Options panel (Figure 4-40), specify the job to watch. By default, you specify to start a PEX data collection by selecting the appropriate radio button. Click **Next**.

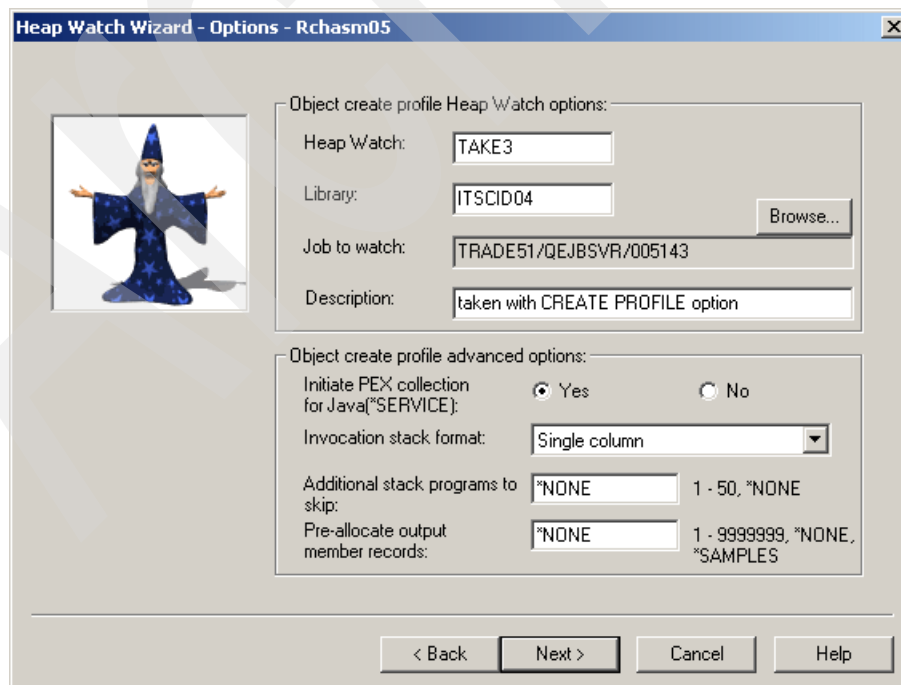


Figure 4-40 Heap Watch Wizard Options panel

3. The Execution Limit panel (Figure 4-41) is displayed. Specify the desired time limit. Specifying a long collection time may cause a noticeable increase in the disk space utilization. You may also specify to use either disk space or the number of samples as a collection limit value. Click **Next**.

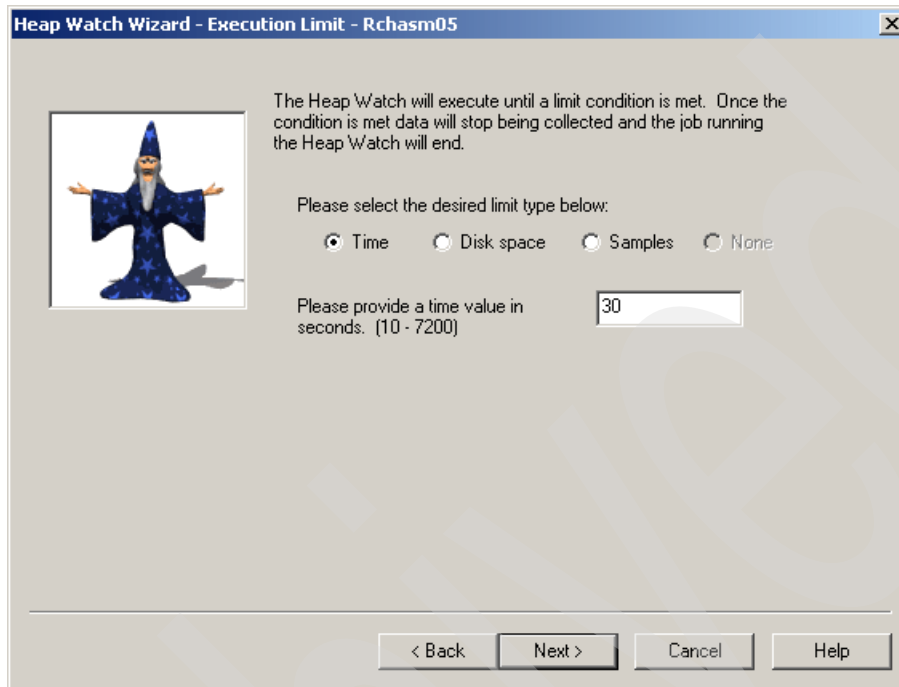


Figure 4-41 Specify when to end the object create collection

4. The Finish panel (Figure 4-42) opens. Scroll down to see the remote command string to be submitted to the server.

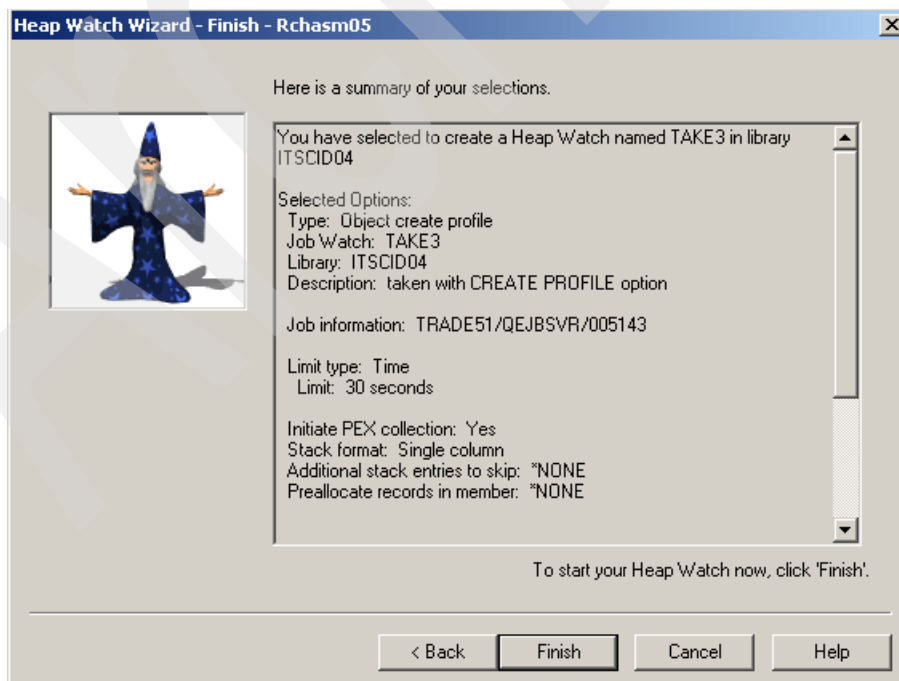


Figure 4-42 Summary panel for object create profile

In this example the command string is:

```
QSYS/SBMJOB CMD(QPYRTJW/WCHJVA ACTION(*OBJCRTPROFILE) OUTMBR(TAKE3) OUTLIB(ITSCID04)  
JOB(018196/QEJBSVR/TRADE51) LIMITYPE(*TIME) TIME(30) STACKFMT(*SINGLECOLUMN)  
STACKSKIP(*NONE) PEXTRIGGER(*YES) PREALLOC(*NONE) TEXT('taken with CREATE PROFILE  
option') ) JOB(WCHJVAPROF) JOBD(QPYRTJW/QPYJWJOB) JOBQ(/) INLLIBL(*JOBDD)  
RTGDTA('QPYJWJOB ITSCID04 TAKE3 ')
```

Click **Finish**.

Collecting for object root finder

To use the object root finder, follow these steps:

1. Return to your object table dump (see Figure 4-43). Right-click **Object counts/sizes by class loader** and select **Open Table(s)**.

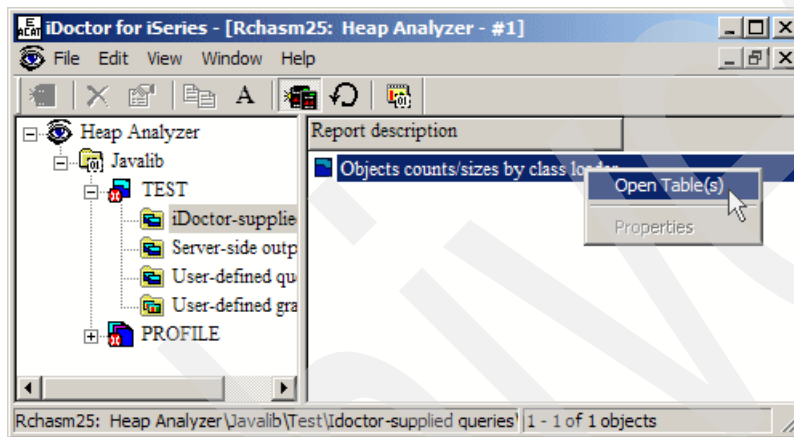


Figure 4-43 *Objects counts/sizes by class loader report*

2. In the new window (Figure 4-44) that opens, right-click any object of interest and select **Start root finder analysis**.

With this support, intense processing is performed on the heap to investigate how the objects are being used. For this operation to work, it has to be done on an active JVM. In fact, any analysis for the root finder must be done when the JVM is active. This differs from the object table dump and object create profile in that, with those tools, you can collect data on an active JVM, and then perform analysis on that data after the JVM is gone.

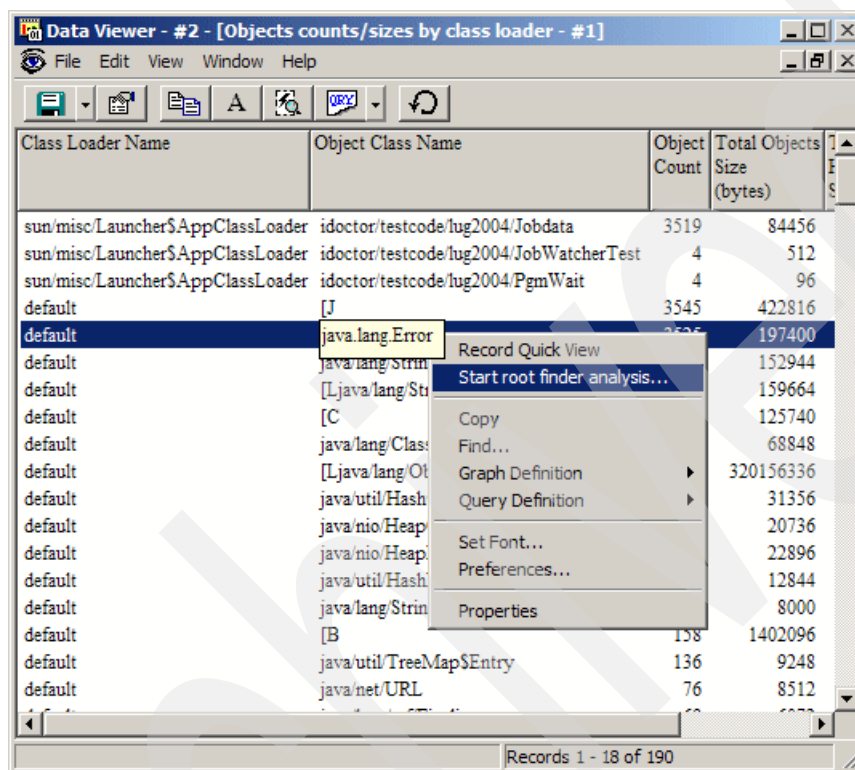
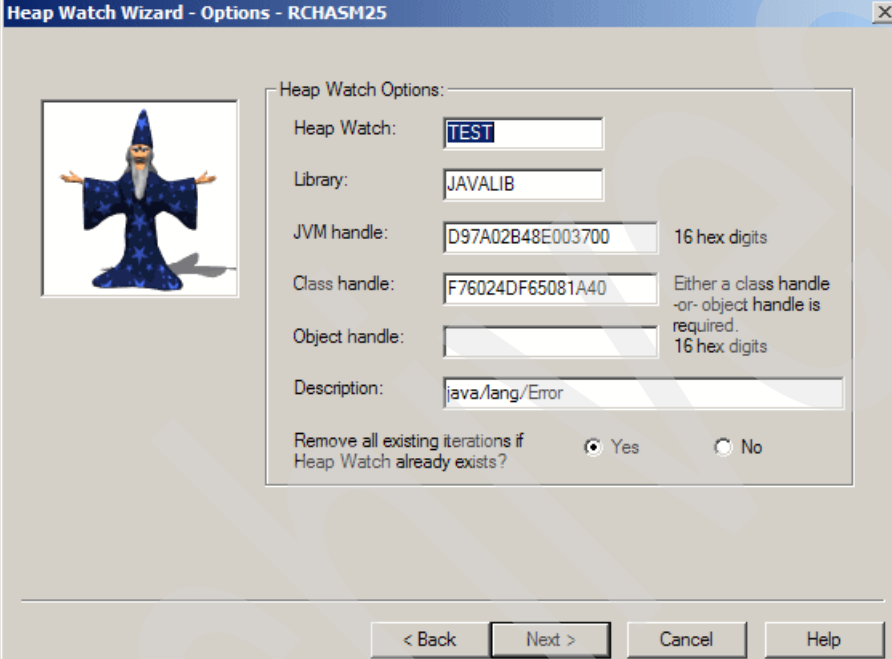


Figure 4-44 Start root finder analysis

3. The Options panel (Figure 4-45) opens, showing the correct information filled in for JVM and Class handle. The only time you have an object handle instead of a class handle is if you choose to run the root finder against a root finder collection. For instance, if you run root finder by selecting a class, the code randomly selects an individual object to investigate. From there, you may find that object A is anchored in object B. You can then select to run the root analysis specifically on object B. In this case, your options display has JVM and Object handle values filled in instead of JVM and Class handles.

If you run previous Heap Watch iterations, you have the option to remove those.

Click **Next** to continue.



Heap Watch Wizard - Options - RCHASM25

Heap Watch Options:

Heap Watch:

Library:

JVM handle: 16 hex digits

Class handle: Either a class handle -or- object handle is required. 16 hex digits

Object handle:

Description:

Remove all existing iterations if Heap Watch already exists? ☒ Yes ☐ No

< Back Next > Cancel Help

Figure 4-45 Options for root finder analysis

4. The root finder processing is intense but does not involve a lot of disk space overhead. For that reason, the only execution limit available is time as shown in Figure 4-46. Click **Next**.

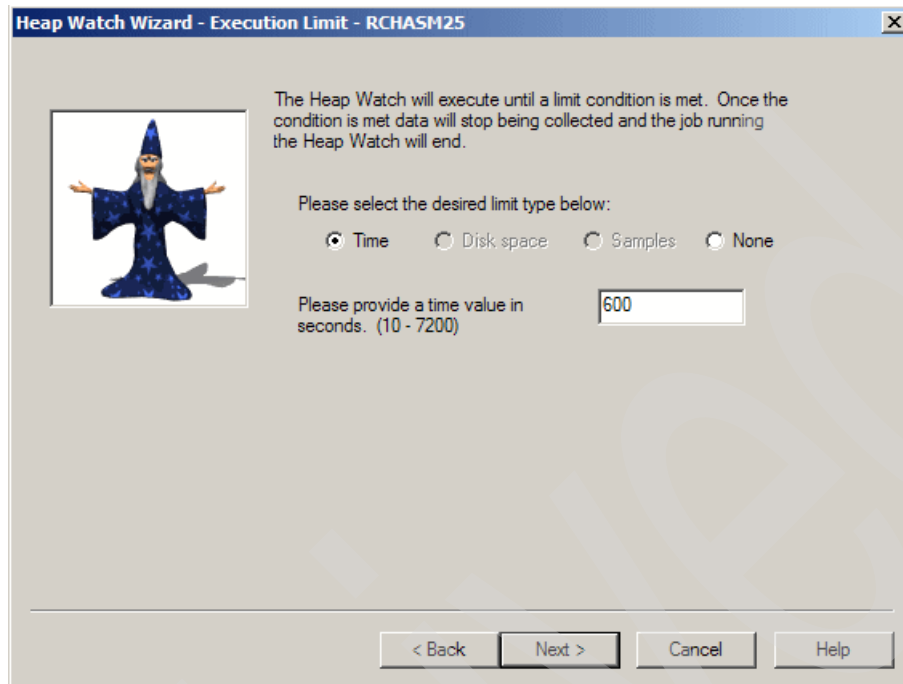


Figure 4-46 Root finder execution limit

5. Review the summary of your selections as shown in Figure 4-47. Click **Finish**.

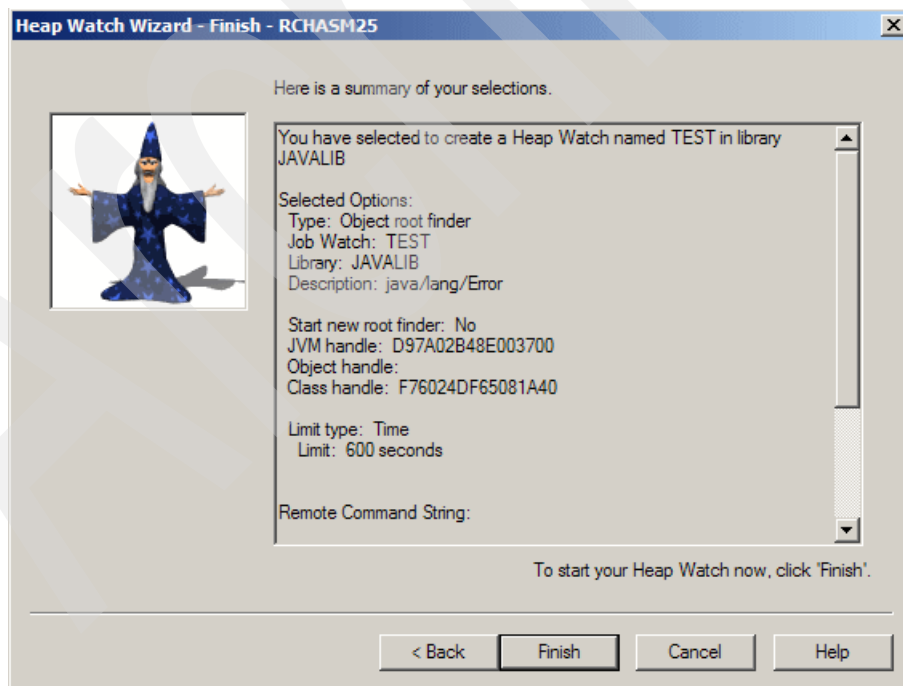


Figure 4-47 Root finder summary

- When the Object Root Finder collection is complete, double-click it to open the tree view of the data displayed in the right pane. Right-click **Object trees by iteration** and select **Explore** as shown in Figure 4-48.

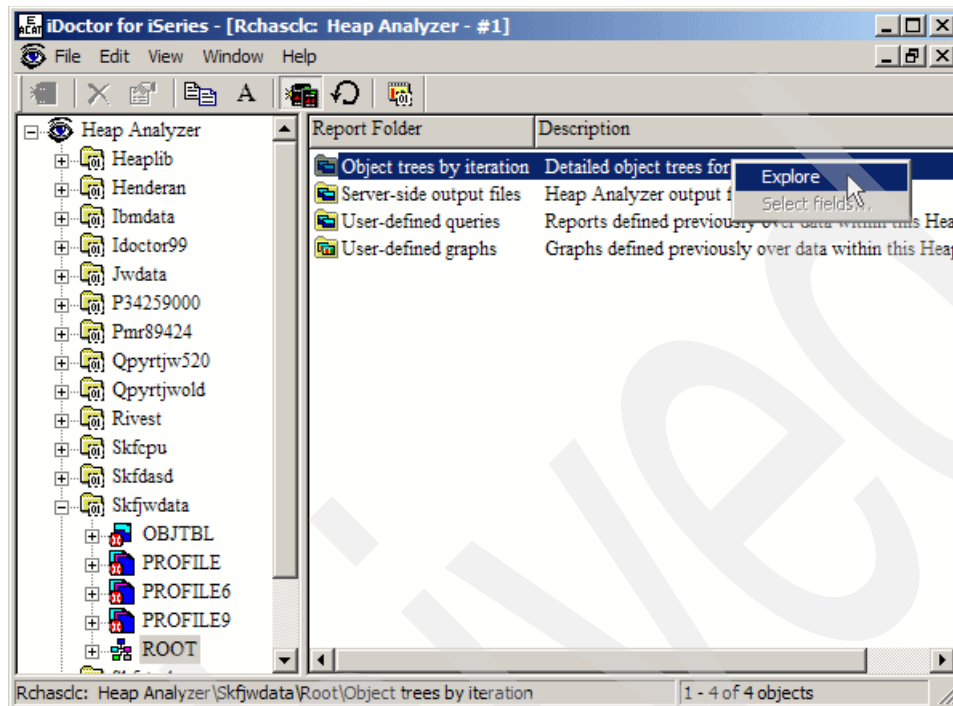


Figure 4-48 Explore object trees by iteration

- Figure 4-49 shows the iteration. Right-click the iteration and select **Open Tree(s)**.

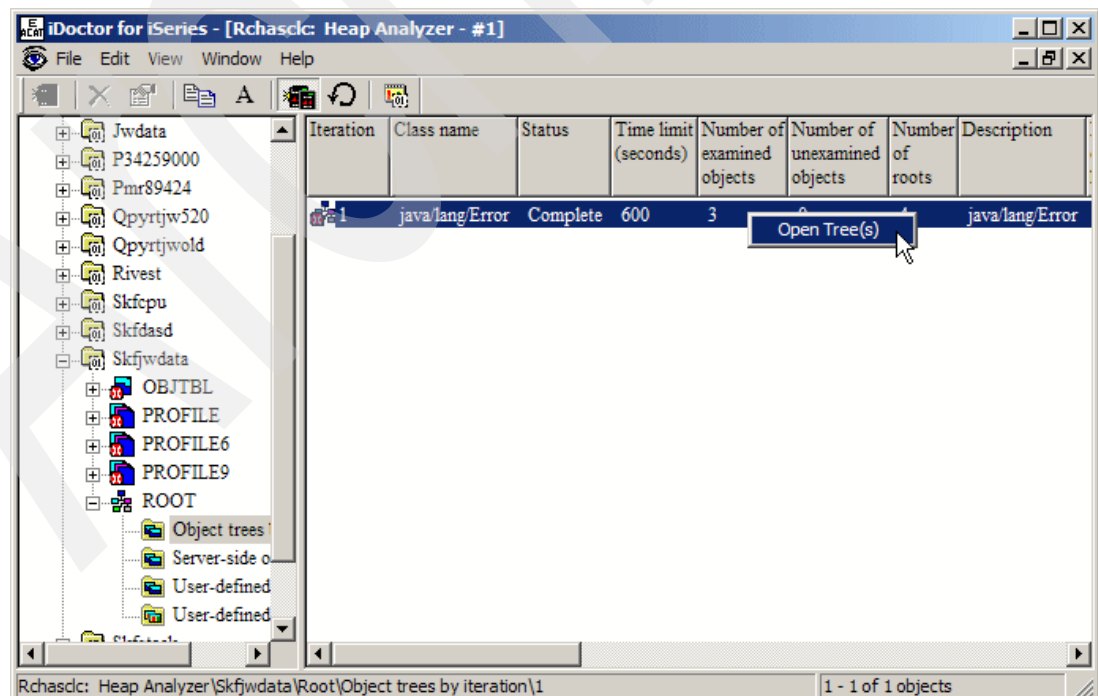


Figure 4-49 Open object trees by iteration

The initial object tree view before expansion is shown in Figure 4-50. Fully expand the tree on the left to see the relationship or relationships and the roots.

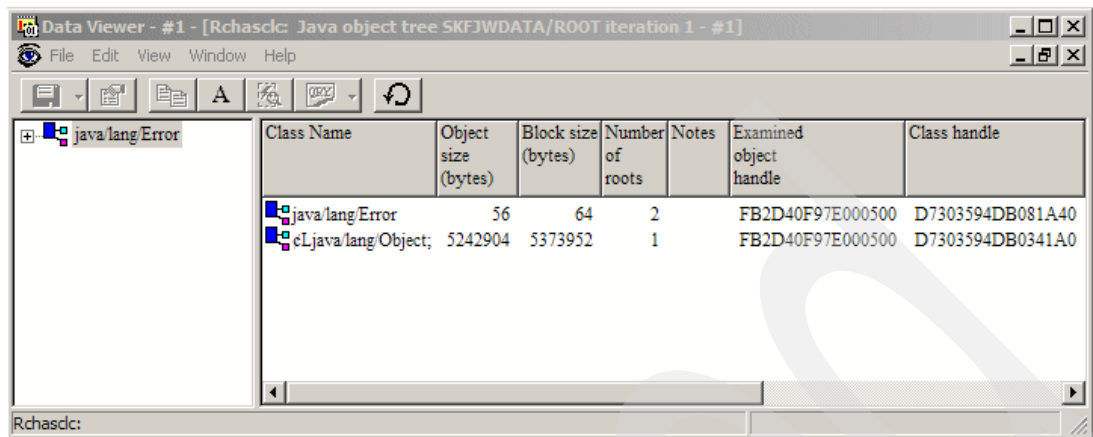


Figure 4-50 Java object tree root iteration before expansion

Figure 4-51 shows the expanded list of objects and roots. From this example, you see that the java/lang/Error object is anchored in a vector.

Because the application is placing them into a vector, they are not getting cleaned by the garbage collector. For any further investigation, you have to go to the application source code or a developer.

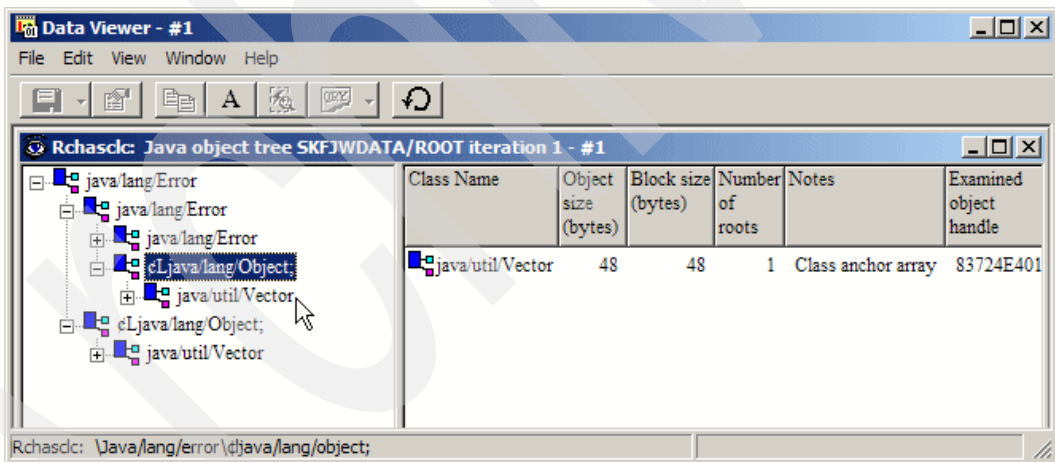


Figure 4-51 Expanded Java object tree root

Creating graphs

When you begin analyzing a performance problem, using a proper data and format to display that data can be a big differentiator. iDoctor tools come with the extensive support of the different types of graphs. This section shows one example of creating a graph based on the data collected by Heap Analysis Tools for Java.

1. Select which report you're going to work with (see Figure 4-52). This depends on the type of problem. In this example, we select to work with **Objects created per interval profile**.

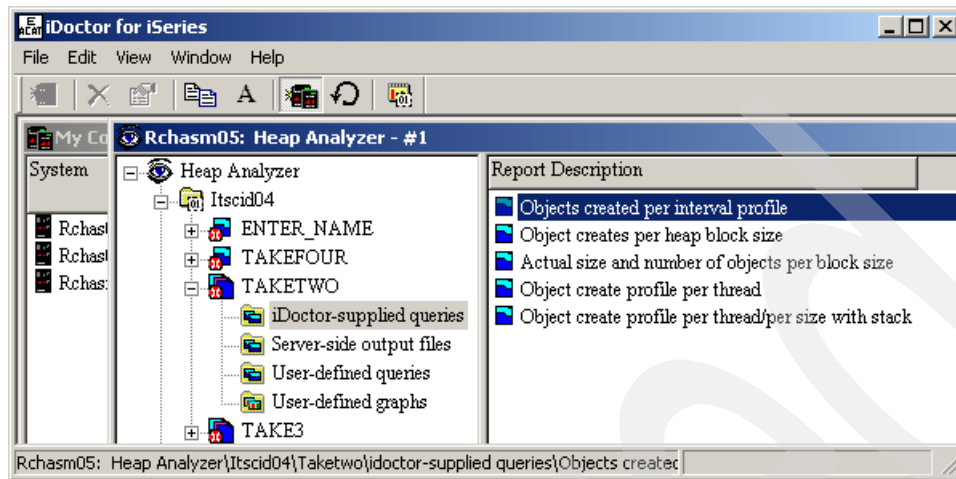


Figure 4-52 Selecting the report

2. When the report window (Figure 4-53) opens, select the time interval which will be your starting point. Right-click the interval and select **Graph Definition** → **Define New**.

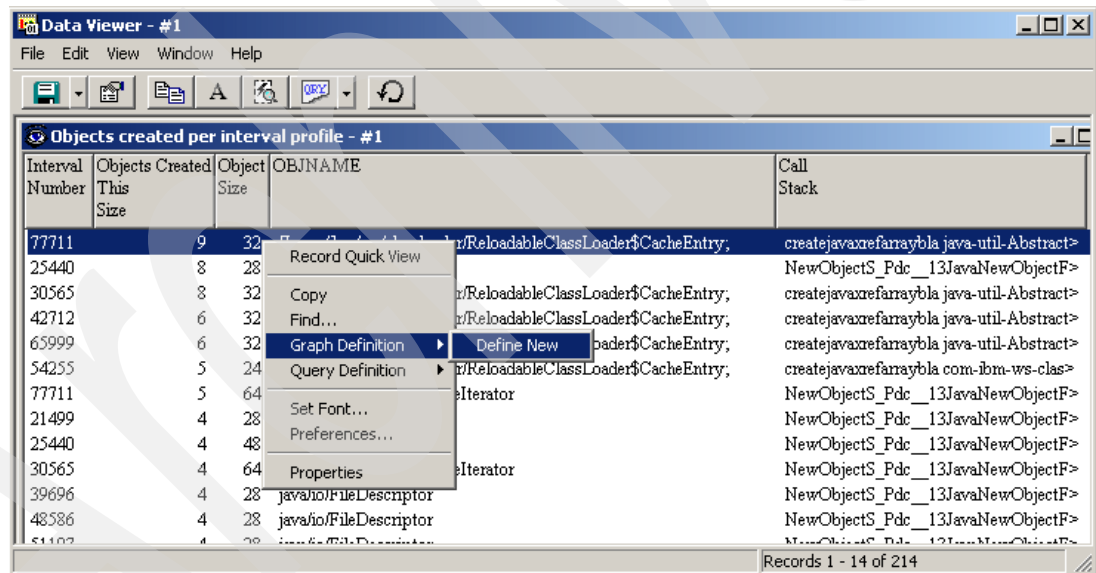


Figure 4-53 Defining a new graph

3. The Graph Definition window (Figure 4-54) opens.
 - a. On the General/X-axis page, enter the name of the graph and select the type of graph that you want to create.

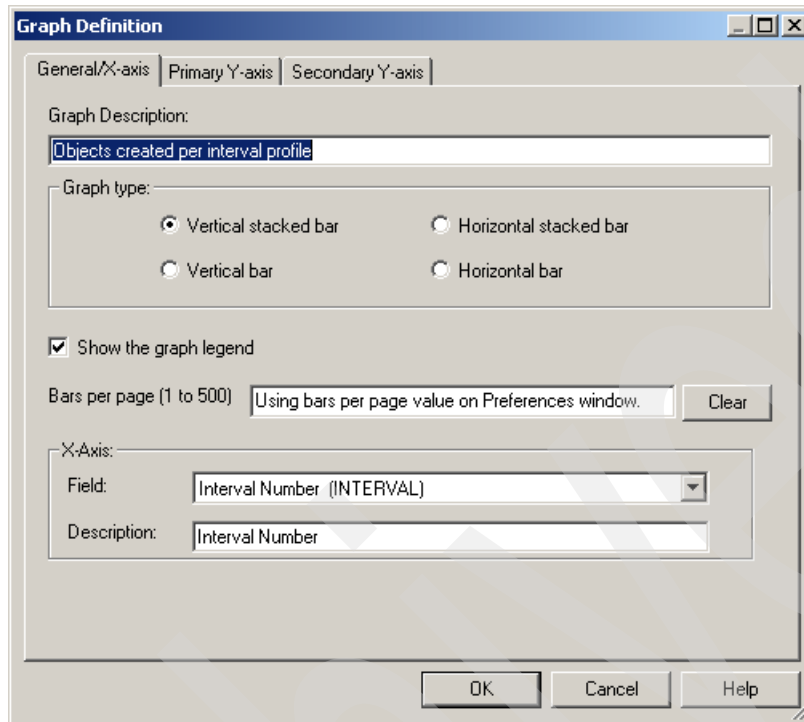


Figure 4-54 Specify what to show on the X-axis

- b. Click the **Primary Y-axis** tab.
 - i. Enter the description and select the parameter for the X-axis. In our case, we select **Object Created This Size** (see Figure 4-55).
 - ii. Click **Add Field**. You can add more than one parameter, which is what we do in Figure 4-58 on page 73. We select to display the object size per interval graph along with the object count.
 - iii. Click **OK**.

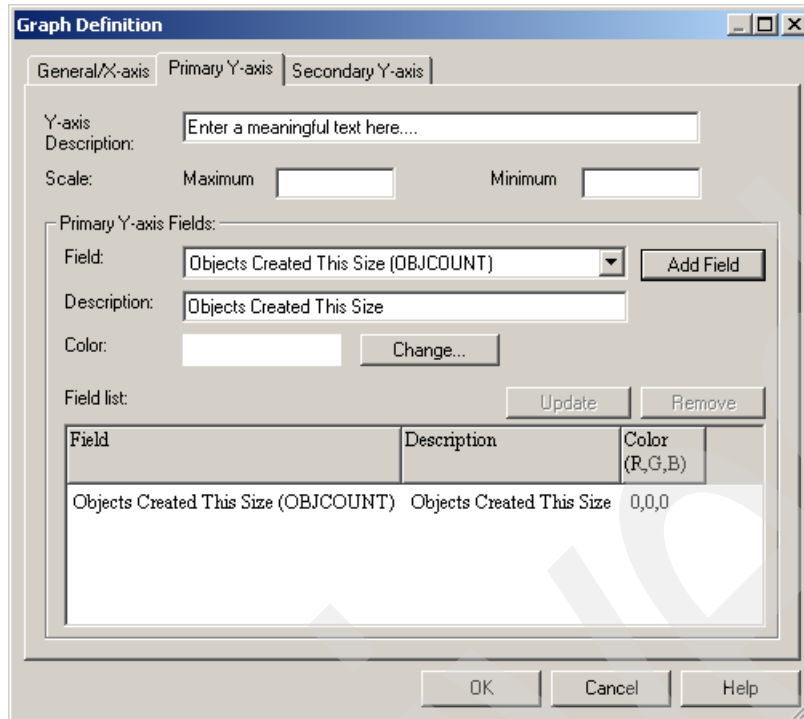


Figure 4-55 Specify what to show on the Y-axis

4. The graph window opens as shown in Figure 4-56. You can hover over any bar to see information for that specific interval.

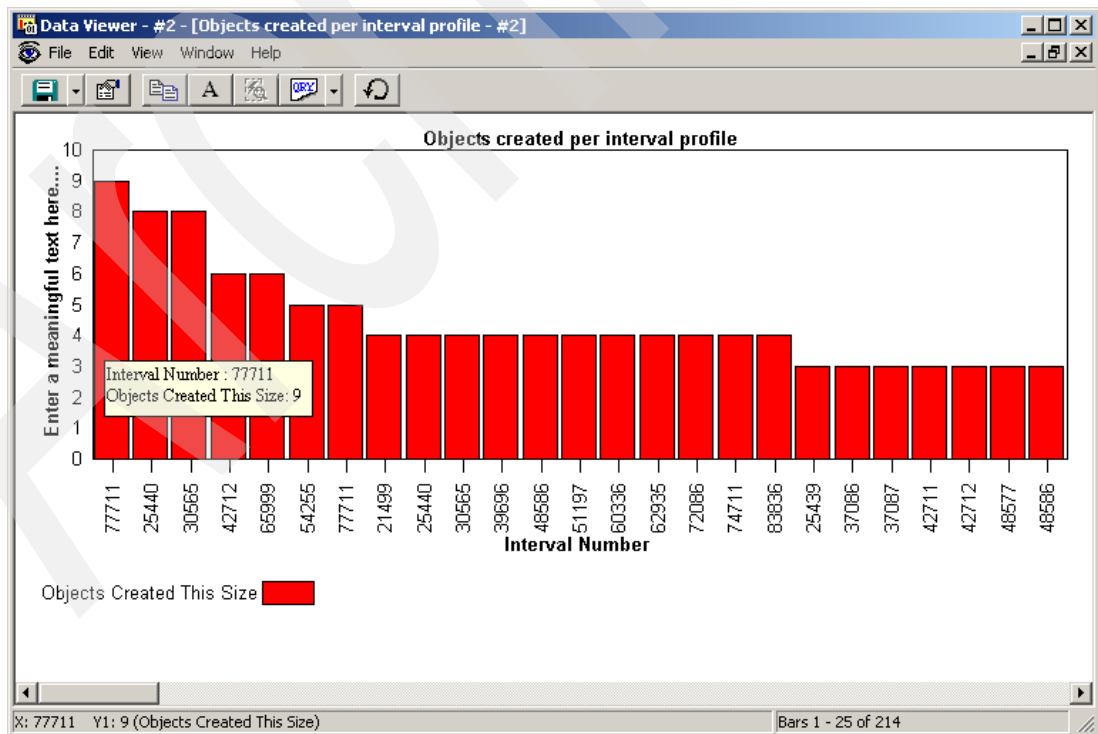


Figure 4-56 Hovering over the graph

If you double-click any bar, the tool opens a new window with the details for that specific time interval as shown in Figure 4-57.

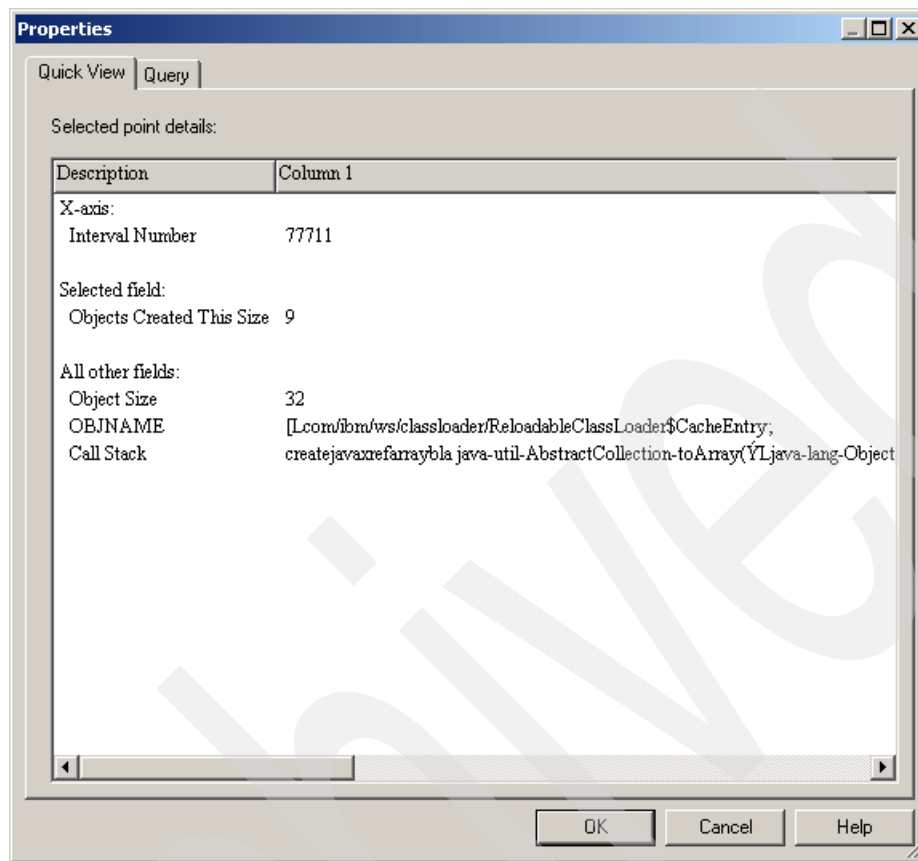


Figure 4-57 Properties window for a time-specific interval

Selecting to display more than one parameter per graph allows you to capture an advanced view of the heap conditions as shown in Figure 4-58.

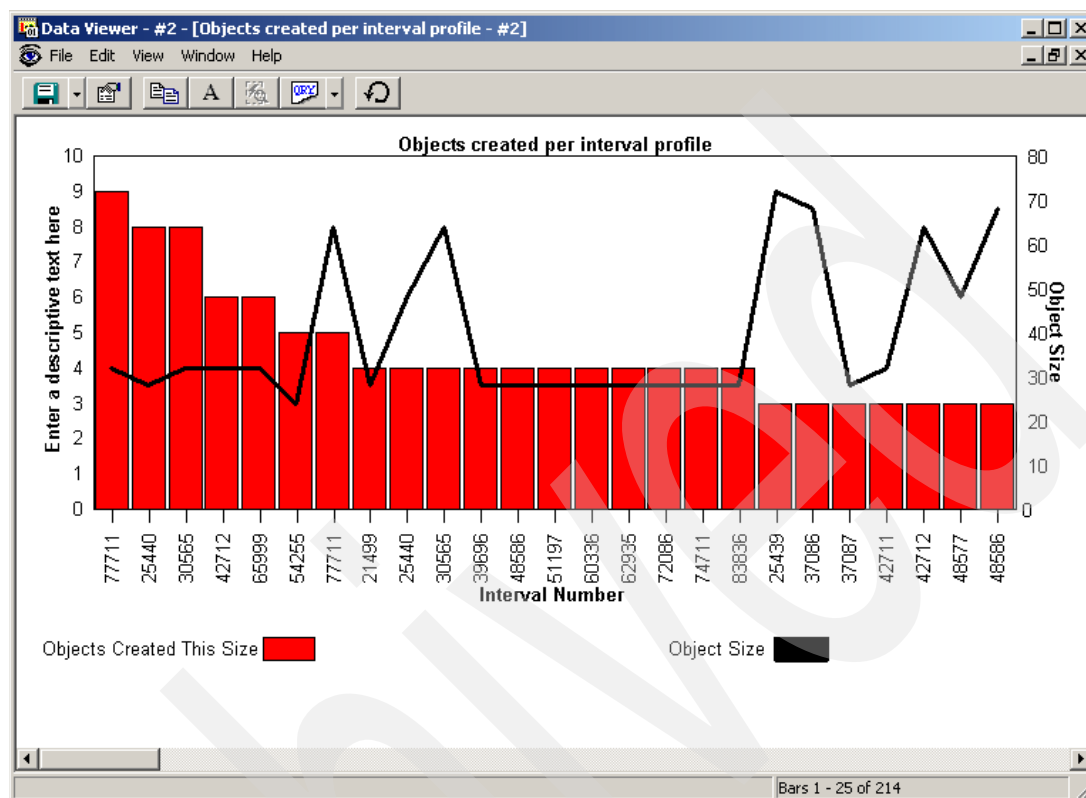


Figure 4-58 Graph with second Y-axis

PEX Analyzer

The iDoctor for iSeries PEX Analyzer Service includes a software tool that is specifically tuned for pinpointing issues that affect system and application performance. The detailed analysis it provides continues where the PM eServer iSeries and Performance Tools products end and supplies a drill-down capability that provides a low-level summary of:

- ▶ Disk operations
- ▶ CPU utilization
- ▶ File opens
- ▶ Machine interface (MI) programs
- ▶ Wait states
- ▶ DASD space consumption and much more

The client component allows you to condense and graph iSeries PEX trace, statistical, and profile data.

PEX Analyzer provides graphical visualization of PEX collection data. Previously, the Print PEX Report (PRTPEXRPT) command (part of the Performance Tools/400 LPP) was the typical method used to view reports created over PEX data. iDoctor for iSeries is a value-add to PRTPEXRPT by providing more flexible interfaces for viewing potentially large and complex data.

PEX Analyzer analyzes PEX trace data that was previously collected on the current or some other iSeries system. PEX Analyzer can analyze only data that has been collected. For example, if the PEX TRACE collection did not include database file activity, PEX Analyzer

cannot produce reports on database file activity. It is important to understand that it is at the time of PEX data collection that the decision is made concerning which types of data to collect. The decision about what data is collected controls the type of analyses that are available.

First, you need to collect the data. This is done by using the PEX Collection Wizard.

1. To begin, right-click a library and select **Create PEX Collection** as shown in Figure 4-59.

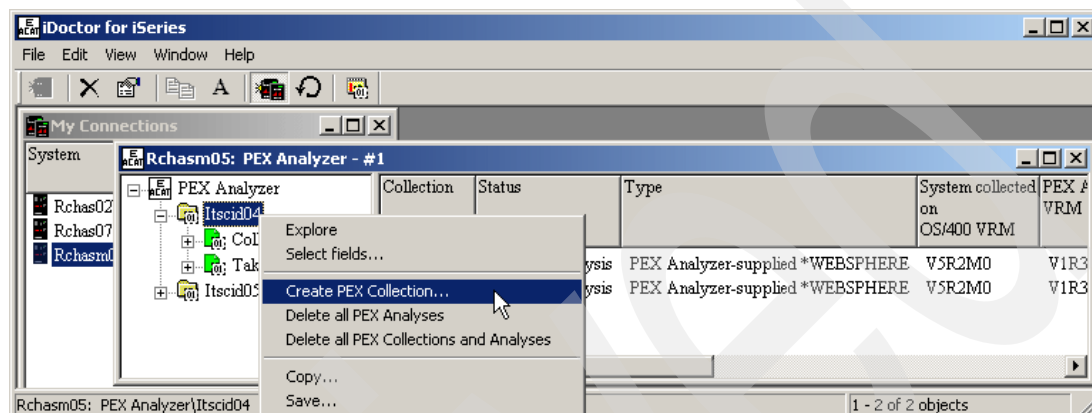


Figure 4-59 Starting a PEX collection

2. The wizard begins. On the Welcome panel (Figure 4-60), you decide which route to take. If you are an advanced user, are familiar with the functions of PEX, and have your own definitions, select **Advanced**. In our example, we choose a simpler path and select **Basic**. Click **Next**.

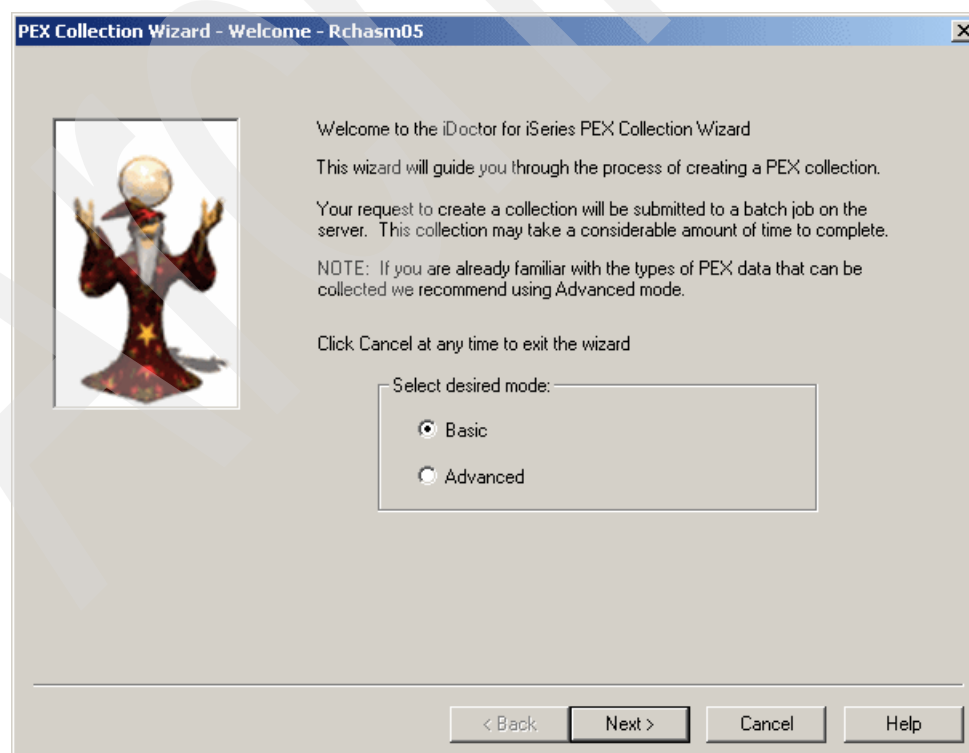


Figure 4-60 PEX Collection Wizard

3. The Problem Type Selection display (Figure 4-61) opens. It asks about the type of the problem. In this example, we select the **WebSphere application(s) is running poorly** option. Click **Next**.

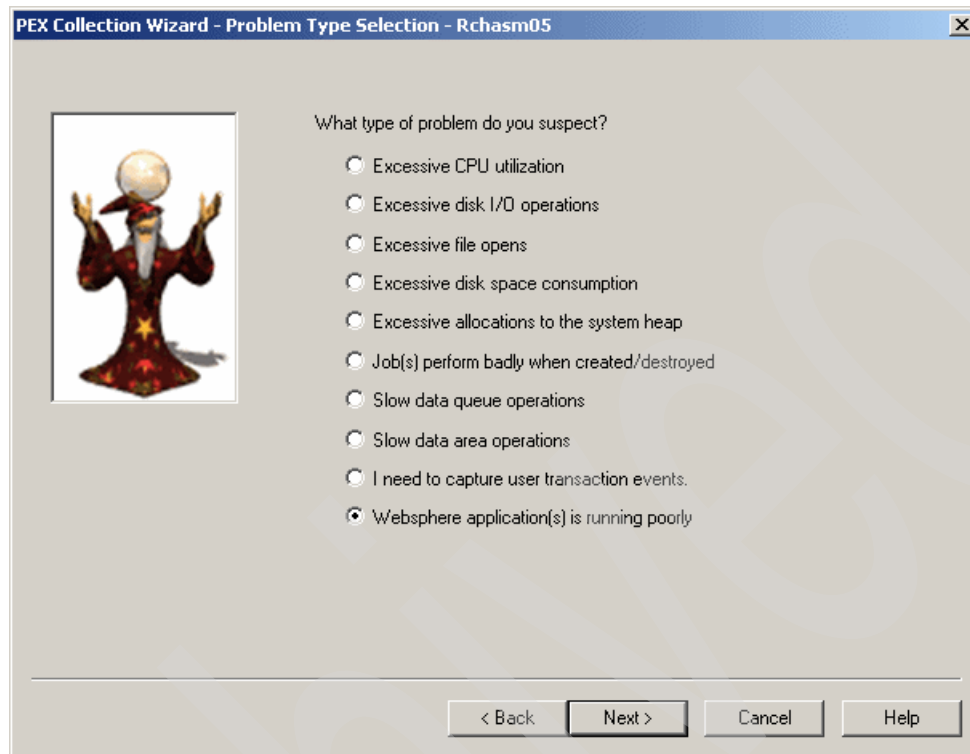
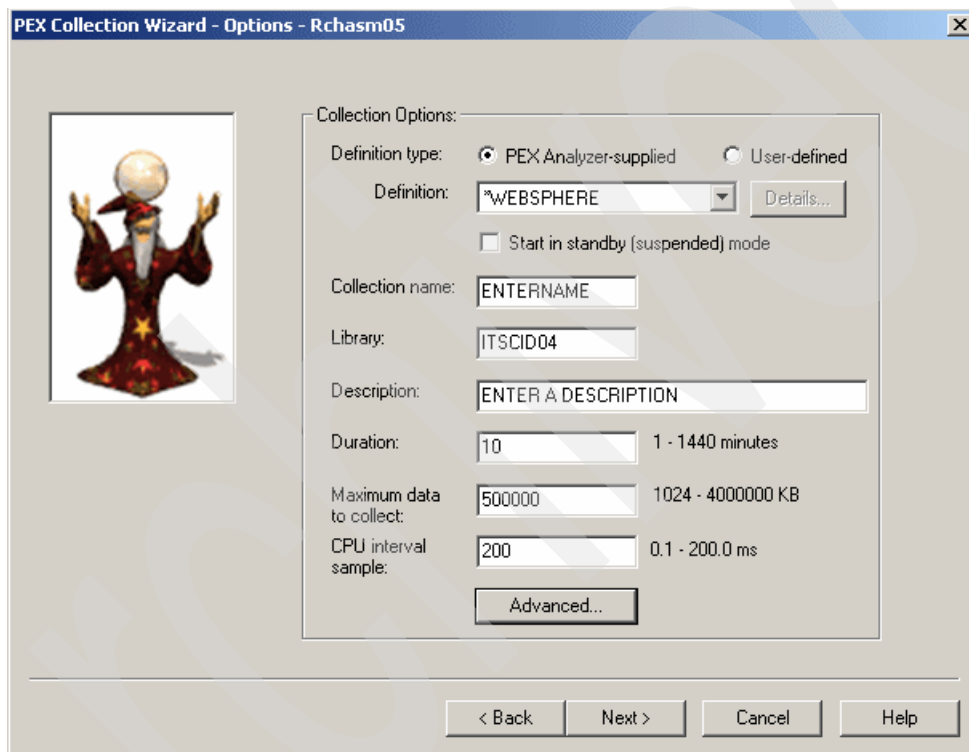


Figure 4-61 Specifying what to collect

4. The Options panel (Figure 4-62) opens.
 - a. Based on your selection in the Problem Type Selection panel, the wizard chooses the PEX definition that fits your problem. In our example, the name of the definition is *WEBSPPHERE. You can select a different one from the list of the predefined definitions.
 - b. If you have experience with PEX, you can select the **User-defined** option for the definition selection and provide your own definition.
 - c. Change the Collection name parameter.
 - d. Optionally, select different values for the duration of the collection, the amount of space allocated for the collected data, or both.
 - e. Click **Next**.



The image shows a Windows-style dialog box titled "PEX Collection Wizard - Options - Rchasm05". On the left is a small icon of a wizard. The main area is titled "Collection Options:" and contains several settings:

- Definition type:** Two radio buttons. "PEX Analyzer-supplied" is selected, and "User-defined" is unselected.
- Definition:** A dropdown menu showing "*WEBSPPHERE" and a "Details..." button.
- Start in standby (suspended) mode:** An unchecked checkbox.
- Collection name:** A text box containing "ENTERNAME".
- Library:** A text box containing "ITSCID04".
- Description:** A text box containing "ENTER A DESCRIPTION".
- Duration:** A text box with "10" and a range "1 - 1440 minutes".
- Maximum data to collect:** A text box with "500000" and a range "1024 - 4000000 KB".
- CPU interval sample:** A text box with "200" and a range "0.1 - 200.0 ms".
- Advanced...:** A button.

At the bottom are four buttons: "< Back", "Next >", "Cancel", and "Help".

Figure 4-62 Specifying the collection options

5. The Trace Additional Events panel (Figure 4-63) enables you to include more events to collect. Your selection depend on the nature of the problem. Click **Next**.

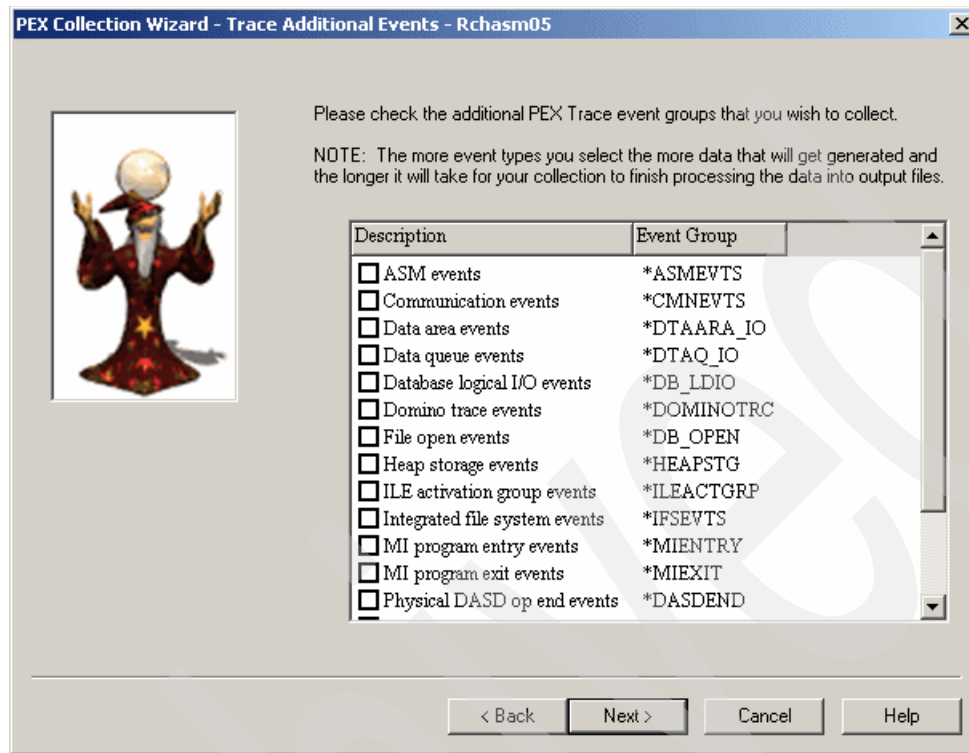


Figure 4-63 Choosing additional events to collect

6. In the Job/Task Option panel (Figure 4-64), select which jobs or tasks to monitor: all jobs or tasks or a specific one. In this example, we select **All jobs**. However, if you identify that the problem is within WebSphere, it's better to select a specific job to reduce the amount of information collected by PEX.

Click **Next**.

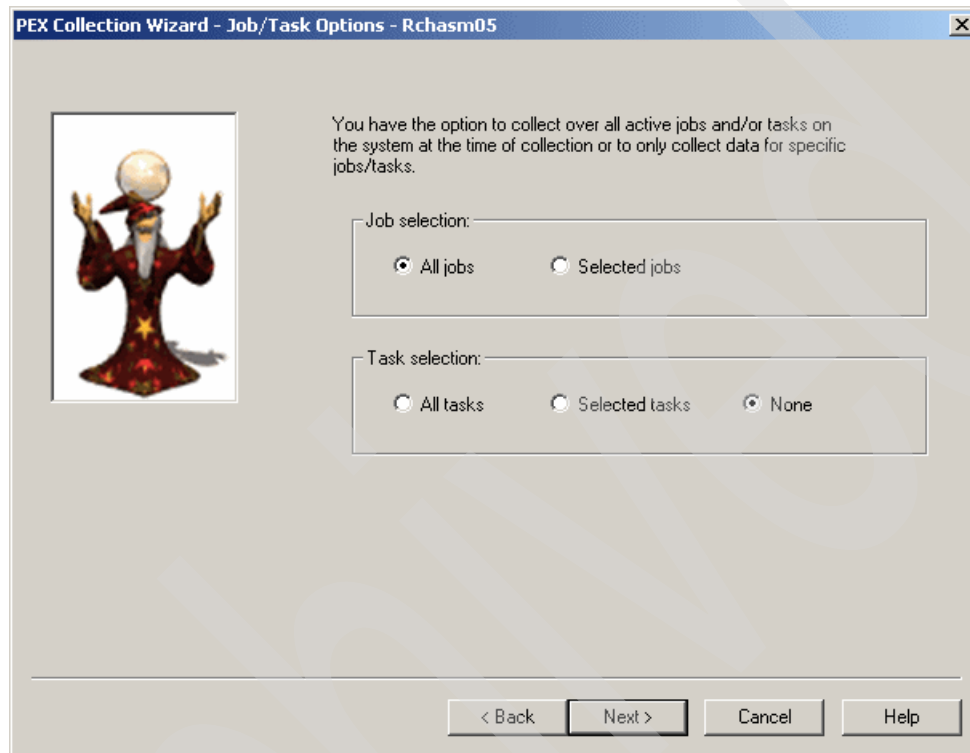


Figure 4-64 Selecting all jobs versus a subset of jobs

7. In the Summary panel (Figure 4-65), review the summary of your selections and click **Finish**.

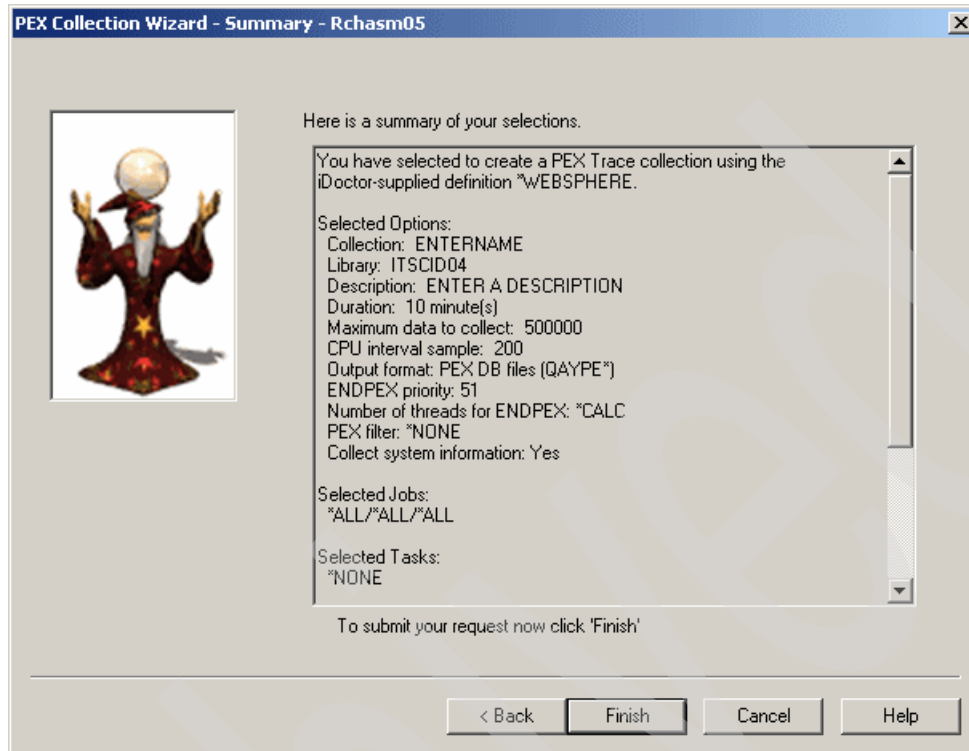


Figure 4-65 PEX collection summary

You can see the status of your collection in the PEX Analyzer window (Figure 4-66).

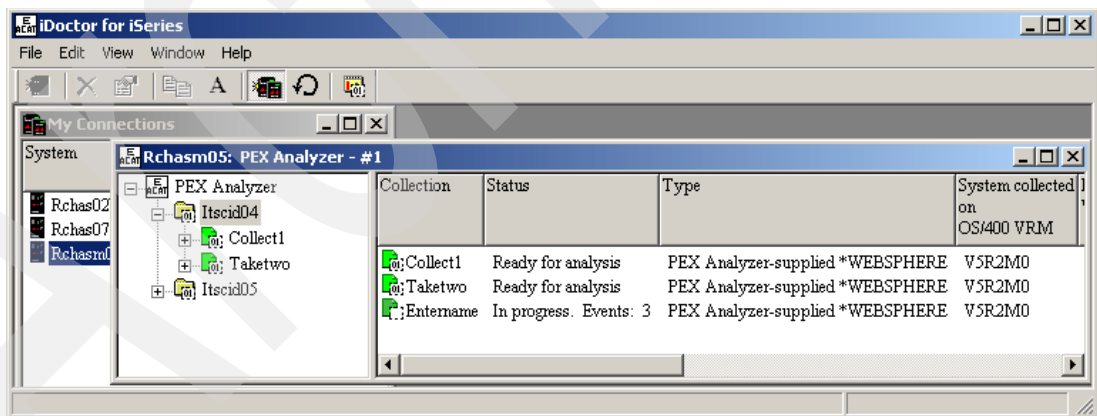


Figure 4-66 PEX data collections panel

After the collection finishes, right-click the collection in the tree and select one of the available analysis options:

- ▶ **Start Analysis Wizard:** This option takes you through several panels where you decide which type of analysis you want to perform.
- ▶ **Run default 'CPU' analysis:** This option produce a report that summarize information related to the CPU activity on the system
- ▶ **Run default 'GENERAL' analysis:** We select this option as illustrated in Figure 4-67.

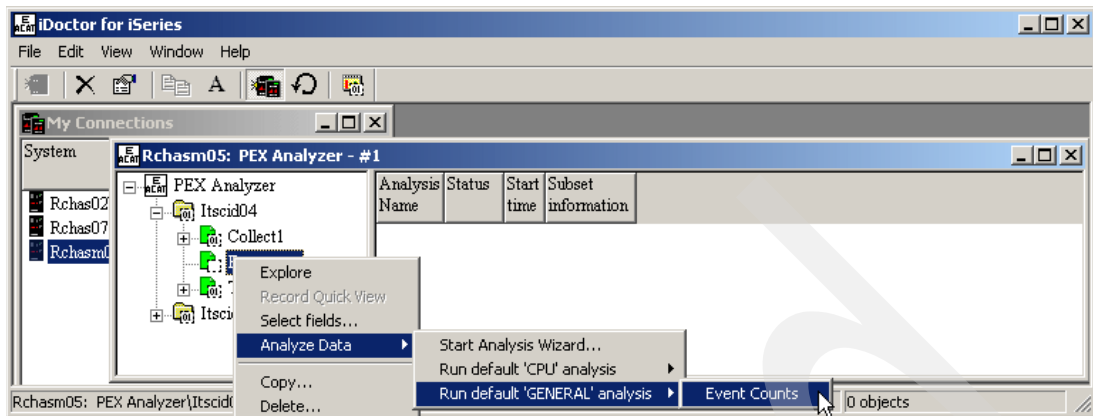


Figure 4-67 Analyzing the PEX data

You should see a new analysis added to your collection (see Figure 4-68).

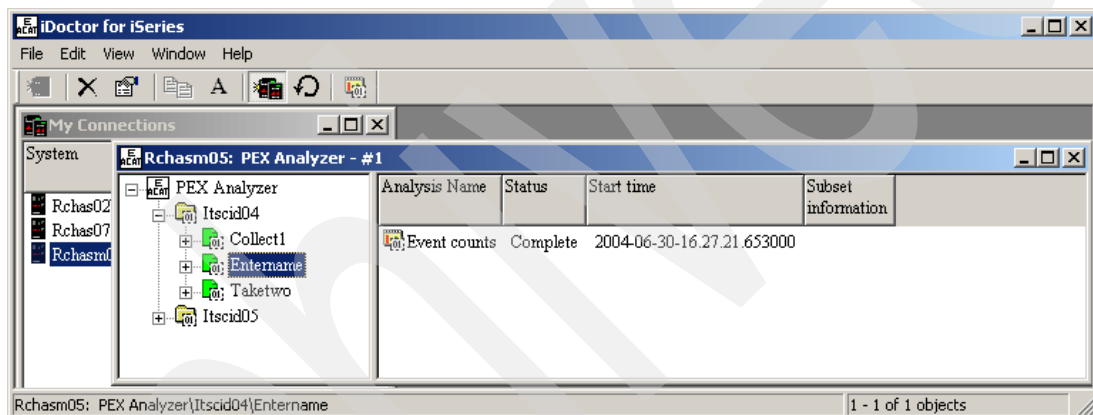


Figure 4-68 Generating the analysis

If you expand your analysis, you see several types of reports that you can review to identify the problem (see Figure 4-69).

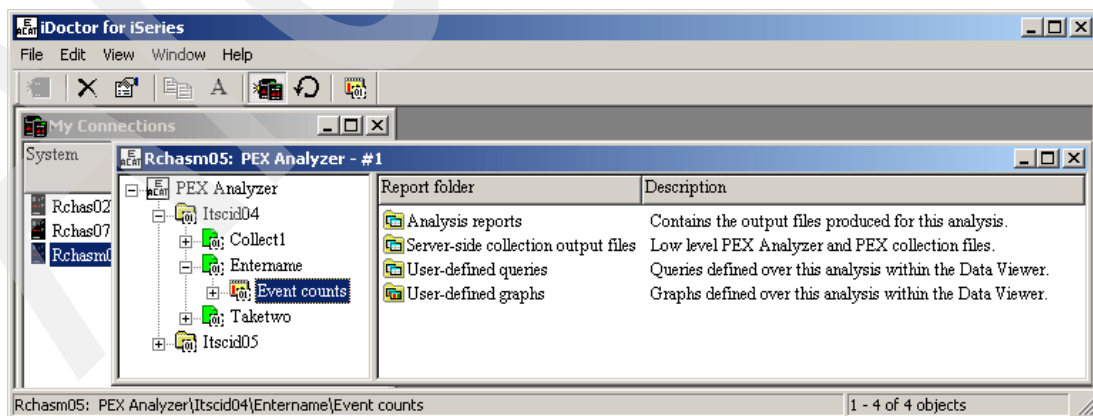


Figure 4-69 Selecting the report to view

4.2.7 HEAPANA command

In 4.2.6, “iDoctor” on page 40, you learned about iDoctor and how you can use its tools, such as Job Watcher and Heap Analysis Tools for Java, to gain an in depth understanding of your application. IBM also supplies an operating system command-line interface with iDoctor that helps manage the collection of data for heap-related issues. The command is called HEAPANA.

HEAPANA handles the execution of the Table Dumps, Profiling, and Job Watcher collections described previously in 4.2.6, “iDoctor” on page 40. It provides a single interface, which simplifies your collection activities. In addition, HEAPANA ensures that the data you collect using the various iDoctor tools captures the exact same time period. This enables you to move easily between data derived from different tools as you research performance problems.

Usage

The HEAPANA command is shipped with Job Watcher. To execute the HEAPANA command, you need to add the QPYRTJW library to the library list and simply type HEAPANA on an iSeries command line. Press F4 to view the parameters for the command, as illustrated in Figure 4-70.

```
Heap Analysis. (HEAPANA)

Type choices, press Enter.

JVM Job Name . . . . . > SERVER1
JVM Job User . . . . . > QEJBSVR
JVM Job Number . . . . . > 012243
Run as Trigger ? . . . . . YES
Library to save data to . . . . > ITSCIDXXHA
Interval(secs between samples) > 60
Samples (# of collections) . . . > 30
Output Queue . . . . . PAGESTATS
Job Watcher Intervals to get . . > 10
Profiler Timer (Seconds) . . . . > 30
```

Figure 4-70 HEAPANA: Executing the HEAPANA command

The available parameters for HEAPANA are explained here. To see all parameters, press F9.

- ▶ **JVM Job Name, JVM Job User, JVM Job Number:** The qualified job name of the JVM.
- ▶ **Run as Trigger?:** This parameter indicates whether you intend to use Job Watcher trigger support to monitor the heap and then trigger the collection of additional information. For details on the Job Watch trigger support, refer to “Job Watcher trigger support” on page 85.
- ▶ **Library to save data to:** This is the library where the collected data is saved. You can name it as you choose. If it doesn’t exist, it is created.
- ▶ **Interval:** This parameter is used to determine how often snapshots of the heap are taken. The “right” value for this parameter depends on a few factors, such as how long you intend the collection to run and how rapid the growth is. You need to ensure that enough snapshots are taken so that you can have a feel for the growth that is occurring.
- ▶ **Samples:** This indicates the number of snapshots that will be collected.

- ▶ **Output Queue:** The output queue input on this parameter is created in the library provided in the Library parameter. You can name the output queue as you like. Snapshots of system status information are written to this output queue during the analysis.
- ▶ **Job Watcher Intervals to get:** Job Watcher is setup to collect x number of intervals as fast as possible. You use this parameter to define what x is going to be. When determining what x should be, you must be concerned with the amount of DASD required to store the collected data. It is not uncommon for collected data to consume 3 GB to 5 GB of storage or more. Therefore, on systems where free storage is at a premium, you need to adjust this parameter accordingly. Also remove the library that contains the collected data after your problem research is completed.
- ▶ **Profiler Timer:** This parameter determines how long you want object create profiler to run. We suggest that you match the profiler timer with the heap dumps setting. For example, if you set the parameters to collect heap dumps for 30 minutes, then you may set the profiler to run for the same duration by setting Profile Timer to 1800 seconds.
- ▶ **Run in Batch:** With this parameter, you have the option to run HEAPANA interactively or in batch mode. If you want to run HEAPANA interactively, type NO for this parameter. When running HEAPANA interactively, your session is tied up for the entire collection time. We recommend that you run HEAPANA in batch by specifying YES on this parameter.

Note: You do not see the Run in Batch parameter if you select the Run as Trigger mode. All trigger operations are submitted to batch and run in the QSYSWRK subsystem.

- ▶ **Trigger Rules Library:** This is the library where the file QPYRTJWRD is found. This file contains the rule definitions for this collection. You only see this parameter if you select YES for the Run as Trigger parameter. For more details about Job Watch trigger support, refer to “Job Watcher trigger support” on page 85.
- ▶ **Trigger Rules Member:** This is the actual member name in the file QPYRTJWRD that you should use for the rules for this collection. You only see this parameter if you select YES for the Run as Trigger parameter. For more details about Job Watch trigger support, refer to “Job Watcher trigger support” on page 85.

Aside from our recommendation to run HEAPANA in batch mode, we also find that your collection should start at, or near the start of, the growth and continue into it. When your collection timing is right, the data saved from HEAPANA provides you with the ability to:

- ▶ Identify the objects that are causing the heap growth
- ▶ Identify the affects of the growth on the memory pool or pools
- ▶ Gain a feel for the objects created and the rate at which they are being created
- ▶ Track the operation of the garbage collector, the size of the heap, and so on
- ▶ At a thread level, see which threads are allocating DASD and which stack frames are for them

After you specify the HEAPANA parameters and execute a collection, you can access a variety of information. You may want to begin analyzing the collected data by examining the output queue specified on the HEAPANA command for the snapshots of the system status during the data collection. Figure 4-71 shows an example of the system status output that is written to the output queue.

| System Status Information | | | | | | | | | | |
|------------------------------|---------|----------|-------|--------------|--------------|-------|----------------|-------|------|------|
| 5722SS1 V5R2M0 020719 | | | | | | | | | | |
| % CPU used : | | | | 93.1 | | | System ASP . . | | | |
| % DB capability : | | | | 12.6 | | | % system ASP u | | | |
| Elapsed time : | | | | 00:06:00 | | | Total aux stg | | | |
| Jobs in system : | | | | 2579 | | | Current unprot | | | |
| % perm addresses : | | | | .007 | | | Maximum unprot | | | |
| % temp addresses : | | | | .013 | | | | | | |
| Sys | Pool | Reserved | Max | -----DB----- | ---Non-DB--- | Act- | Wait- | Act- | | |
| Pool | Size M | Size M | Act | Fault | Pages | Fault | Pages | Wait | Inel | Inel |
| 1 | 535.66 | 227.91 | +++++ | .0 | .0 | 1.5 | 1.5 | 16.7 | .0 | .0 |
| 2 | 6007.46 | 2.16 | 1521 | .4 | .9 | 6.9 | 16.7 | 998.7 | .0 | .0 |
| 3 | 3000.00 | .22 | 2585 | .0 | .0 | 1.4 | 9.9 | 8501 | .0 | .0 |
| 4 | 528.76 | .01 | 150 | .0 | .0 | .1 | .2 | 5.9 | .0 | .0 |
| 5 | .25 | .00 | 1 | .0 | .0 | .0 | .0 | .0 | .0 | .0 |

Figure 4-71 HEAPANA System Status Information

Aside from the system status snapshots, HEAPANA saves detailed collection information in a series of files in the library that you specified on the HEAPANA command. Table 4-2 shows the file names and descriptions of the data collected by HEAPANA.

Table 4-2 HEAPANA: Description and location of data collected

| File name | Information collected |
|------------|---|
| QPYRTJVMF | Java Object Create Profiler control file |
| QPYRTJVMF0 | Java Object Create Profiler per interval header |
| QPYRTJVMF1 | Java Object Create Profiler thread information |
| QPYRTJVMF3 | Java Object Create Profiler data, stack fmt 2 |
| QPYRTJVMH0 | Java Object Table Dump control file |
| QPYRTJVMH1 | Java Object Table Dump class loader information |
| QPYRTJVMH2 | Java Object Table Dump object information |
| QPYRTJWA | Basic job statistics |
| QPYRTJWAAI | Activation group/program activation statistics |
| QPYRTJWAC1 | Communications socket and IP |
| QPYRTJWAC2 | Jobs attached to a communications socket |
| QPYRTJWAD | Wait bucket descriptions |
| QPYRTJWAJ1 | Java thread wait statistics |
| QPYRTJWAJ2 | Java JVM statistics |
| QPYRTJWAMT | Missed threads |
| QPYRTJWAPR | Per-Process statistics |
| QPYRTJWARI | Job-specific Job Watch control file |
| QPYRTJWASD | SQL statement host variables |
| QPYRTJWASQ | SQL statement and package information |

| File name | Information collected |
|------------|---|
| QPYRTJWAST | Invocation stack |
| QPYRTJWAS3 | SQL OCL and Prepared Statement Array (PSA) overview |
| QPYRTJWAS4 | SQL Open Cursor List Array (OCL) |
| QPYRTJWAS5 | SQL PSA |
| QPYRTJWAT2 | Wait enum descriptions |
| QPYRTJWA1T | Task/thread first time information |
| QPYRTJWSYS | Job Watcher static system information file |

With the collected data in place, you can view the structure of these files and their contents to obtain useful information to resolve performance issues. For example, Figure 4-72 shows the structure of the QPYRTJVMH2 file, which contains Object Table Dump object information. As you can see, you can capture useful information, such as the object count, size, and class name.

| Table | Columns | Key Constraints | Foreign Key Constraints | Check Constraints | | |
|-------------|------------|-----------------|-------------------------|-------------------|---------------|-------------------------------|
| Column Name | Short Name | Data Type | Length | Nulla... | Default Value | Text |
| GCCYCLE | GCCYCLE | INTEGER | | No | 0 | GC Cycle |
| LOADERID | LOADERID | INTEGER | | No | 0 | Class Loader ID |
| GLBLOBJECT | GLBLOBJECT | CHARACTER | 1 | No | '' | Global Registry Info |
| GLBLROOTS | GLBLROOTS | CHARACTER | 1 | No | '' | Y = roots in Global Reg. |
| OBJCOUNT | OBJCOUNT | DECIMAL | 20,0 | No | 0 | Object Count |
| TOTOBJSIZ | TOTOBJSIZ | DECIMAL | 20,0 | No | 0 | Total Objects Size (bytes) |
| TOTHEPASIZ | TOTHEPASIZ | DECIMAL | 20,0 | No | 0 | Total Objs Heap Size (bytes) |
| TRUOCNAML | TRUOCNAML | INTEGER | | No | 0 | True Object Class Name Length |
| OCNAME | OCNAME | VARCHAR | 5000 | No | '' | Object Class Name |
| ANOBJHNDL | ANOBJHNDL | CHARACTER | 16 | No | '' | Class Handle |

Figure 4-72 HEAPANA: Viewing the structure of the collected data

Figure 4-73 shows the data that is captured in the QPYRTJVMH2 file.

| | OBJCOUNT | TOTOBJSIZ | TOTHEPASIZ | TRUOCNAML | OCNAME |
|-----|----------|-----------|------------|-----------|---|
| 85 | 1043 | 137676 | 200256 | 39 | com/ibm/ejs/container/EJBMethodInfoImpl |
| 86 | 1584 | 126720 | 126720 | 27 | java/lang/ref/SoftReference |
| 87 | 3500 | 126000 | 168000 | 26 | com/ibm/ejs/j2c/HandleList |
| 88 | 484 | 125840 | 247808 | 42 | com/ibm/ws/Transaction/JTA/TransactionImpl |
| 89 | 580 | 125280 | 148480 | 37 | org/eclipse/emf/ecore/impl/EClassImpl |
| 90 | 750 | 124112 | 169280 | 45 | [Ljava/lang/ThreadLocal\$ThreadLocalMap\$Entry; |
| 91 | 470 | 124080 | 240640 | 31 | com/ibm/rmi/iiop/CDRInputStream |
| 92 | 453 | 121404 | 231936 | 32 | com/ibm/rmi/iiop/CDROutputStream |
| 93 | 834 | 120096 | 160128 | 41 | com/ibm/ws/console/core/item/PropertyItem |
| 94 | 2470 | 118560 | 118560 | 29 | javax/security/auth/Subject\$6 |
| 95 | 2927 | 117080 | 140496 | 45 | com/ibm/rmi/iiop/OutputStreamHook\$HookPutFields |
| 96 | 657 | 115632 | 126144 | 41 | org/eclipse/emf/ecore/impl/EReferenceImpl |
| 97 | 962 | 115440 | 123136 | 46 | com/ibm/ws/LocalTransaction/LocalTranCoordImpl |
| 98 | 8 | 112192 | 114688 | 26 | [Lcom/ibm/ws/cache/Bucket; |
| 99 | 1553 | 111816 | 124240 | 67 | com/ibm/ws/webcontainer/cache/invoication/CacheableInvocati |
| 100 | 1532 | 110304 | 122560 | 60 | org/eclipse/emf/ecore/impl/EObjectImpl\$EPropertiesHolderImpl |
| 101 | 1578 | 107304 | 126240 | 15 | java/net/Socket |
| 102 | 1392 | 105792 | 111360 | 27 | java/net/SocketOutputStream |

Figure 4-73 HEAPANA: Viewing the data collected

Depending on your outlook, there are a variety of ways to view and analyze the captured data. For example, if you prefer, you can use the STRSQL command to query the various HEAPANA data sources. Alternatively, you can use the iSeries Navigator's built-in Database support, as we did, to analyze the data.

Job Watcher trigger support

Trigger support within Job Watcher can be helpful when you want to collect a set of data to investigate the heap growth of any Java application. The information that is contained is intended to serve as a guide for using this trigger support to help collect the right information and to understand the root cause of heap growth.

The timing of the data collection can be difficult to coordinate. The following steps provide information about how to set up and use Job Watcher to automatically monitor for heap growth. Then it can begin a series of collections that can gather data, including garbage collection table dumps, paging information, Job Watcher, and Object Create Flight Recorder when the growth surfaces.

1. Before you begin, you must:
 - a. Have the job information of the JVM you want to monitor.
 - b. Configure the Job Watcher rules file.
 - c. Start the Job Watch using the HEAPANA command.
2. Enter the WRKACTJOB command and select option 5 to locate the job information for the JVM in the same manner as you would for any job on the system.
3. Record the job information so you can enter it in the job parameters of the HEAPANA command.
4. To take advantage of built-in support for automated monitoring and collection, set up the Job Watcher rules file prior to executing HEAPANA command. The default rules definition file is QPYRTJWRD in the QPYRTJW library. We recommend that you do not make any modifications to this file because it is designed to work with Java Heap Analysis. Instead, make a copy of this file in another library.

In general, you should not need to make any modifications to the majority of the settings in the rules file. The only exception to this is the trigger criteria section and the values that you set. They are unique to each data collection that you execute. In particular, you need to decide the size of the heap you want to monitor for and when you want to start data collection.

The parameters that you must be concerned with include:

- ▶ INITIAL STATE=TRIGGERWAIT
- ▶ TRIGGERWAIT DURATION TIME=xxxxxx, where xxxxx is the number of seconds
- ▶ ACTION=CALL: The possible values are COLLECT, CALL or BOTH
- ▶ CALL=HEAPANA/HEAPANA_TG: The library and program must exist at startup time
- ▶ GCHEAPSZ GT 10000000

Using HEAPANA together with the rules file can help you to more easily manage heap growth and data collection to reduce the time needed to successfully capture and resolve performance concerns.

4.2.8 Database Monitor for iSeries

Database Monitor for iSeries (DB Monitor) is a tool that you can use to analyze database performance problems. While DB Monitor doesn't measure the 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.

We have seen cases where the problem seemed to be in the WebSphere Application Server. Then after a thorough investigation, we found that the problem was caused by a long running SQL statement. After creating a view suggested by running the Visual Explain, as shown in 4.2.10, “SQL Visual Explain” on page 91, the problem was gone.

DB Monitor data is most useful if you have a basic knowledge of iSeries query optimization techniques. Database Monitor has two user interfaces:

- ▶ 5250 commands
 - Start Database Monitor (STRDBMON) command
 - End Database Monitor (ENDDBMON) command
- ▶ iSeries Navigator

Collecting DB Monitor data with 5250 interface

The 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 member. If the file or member does not exist, it is created based on the QAQQDBMN file in the QSYS library.

Enter the STRDBMON command to start the DB Monitor. The DB Monitor collects information about 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.

Attention: When the DB Monitor is gathering data, a significant amount of CPU utilization (20% to 30%) and disk usage may be temporarily required.

STRDBMON parameters

Enter the STRDBMON command and press F4 to see the parameters as shown in Figure 4-74.

Start Database Monitor (STRDBMON)

Type choices, press Enter.

| | | |
|----------------------------------|---------|----------|
| File to receive output | OUTFILE | |
| Library | | *LIBL |
| Output member options: | OUTMBR | |
| Member to receive output . . . | | *FIRST |
| Replace or add records | | *REPLACE |
| Job name | JOB | * |
| User | | |
| Number | | |
| Type of records | TYPE | *SUMMARY |
| Force record write | FRCRCD | *CALC |
| Comment | COMMENT | *BLANK |

Bottom

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

Figure 4-74 Start Database Monitor display

The following list explains the parameters of the STRDBMON command:

- ▶ **OUTFILE:** The file name for the results file is required, but the library name is optional. The file is created if it does not exist. It is 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. In 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 be written 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: The user needs to use the ENDDDBMON *ALL command to end the DB Monitor for all jobs.

ENDDDBMON parameters

The ENDDDBMON parameters and their functions are:

- ▶ **JOB:** The user can specify a particular job name or all jobs (*ALL) to end. 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.

Collecting DB Monitor data using the GUI

Even though the data collected via the GUI is the same as the data collected via the 5250 session, the name is different. The data collected via the GUI is called *SQL performance monitor data*.

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 your system at any time. However, there can only be one monitor instance monitoring all jobs. When collecting information for all jobs, the monitor collects 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 iSeries Navigator, expand **your server** → **Databases** → **your database**.
2. Select **SQL Performance Monitor**, right-click, and select **New** → **Summary** or **Detailed** (see Figure 4-75).

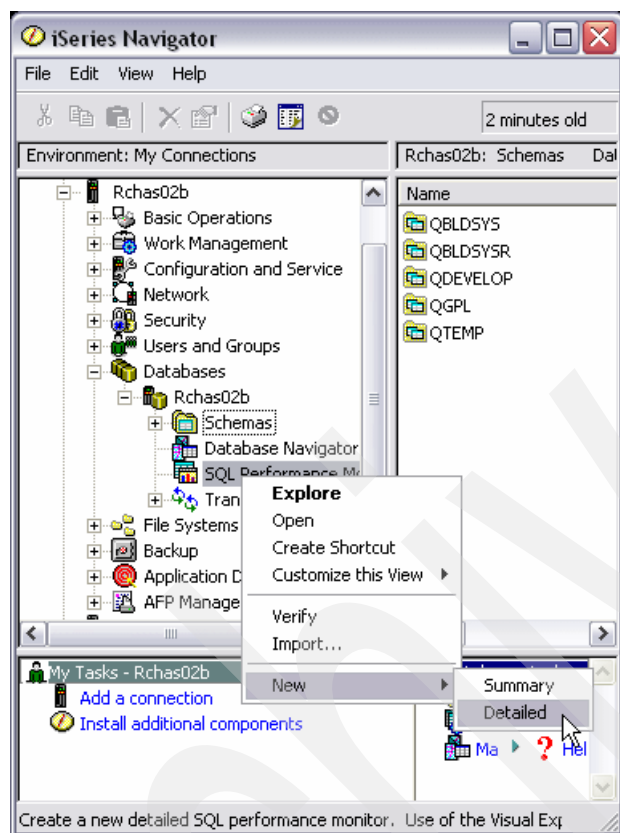


Figure 4-75 Starting to create a new Database Monitor

3. The New Summary SQL Performance Monitor window opens.
 - a. Click the **General** page,
 - i. In the Name field, specify the name you want to give the monitor.
 - ii. In the Library field, specify the library. The collected information is stored in this library.
 - iii. If you want to specify the maximum amount of system memory that the monitor is allowed to consume, in the Storage (MB) field, specify the size in MB.
 - b. Click the **Monitored Jobs** tab.
 - i. 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.
 - ii. 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.
 - iii. If you select 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**.

- c. If you selected Summary monitor, click the **Data to Collect** tab.
 - i. Select the types of data that you want to collect.
 - ii. If you want to collect all types of data, click **Select All**.
 - d. Click **OK**. The performance monitor starts and runs until it is ended or paused.
- If you want to collect overall statistics about the database, select to collect Summary data. The data to be collected is shown in Figure 4-76.

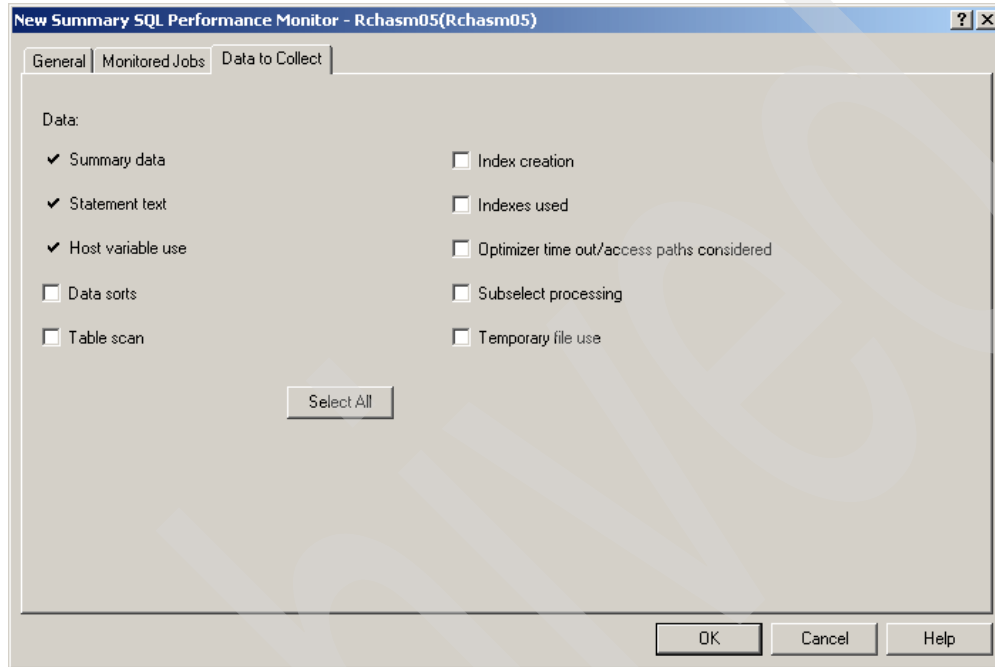


Figure 4-76 Starting to collect summary data with the SQL database monitor

When planning to use the data for tuning the database, select to collect Detailed data. This collects information about:

- Data sorts
- Table scans
- Index creations
- Indexes used
- Optimizer time outs
- Access paths that were considered
- Subselection processing
- Temporary files used

4. Selecting the option to create a new detailed monitor leads you to the display shown in Figure 4-77. Select the job or jobs to monitor and click **OK** as shown in Figure 4-77.

Tip: Click a column heading to arrange the rows in ascending or descending order. You may also drag the column borders to display only the columns that you want to see.

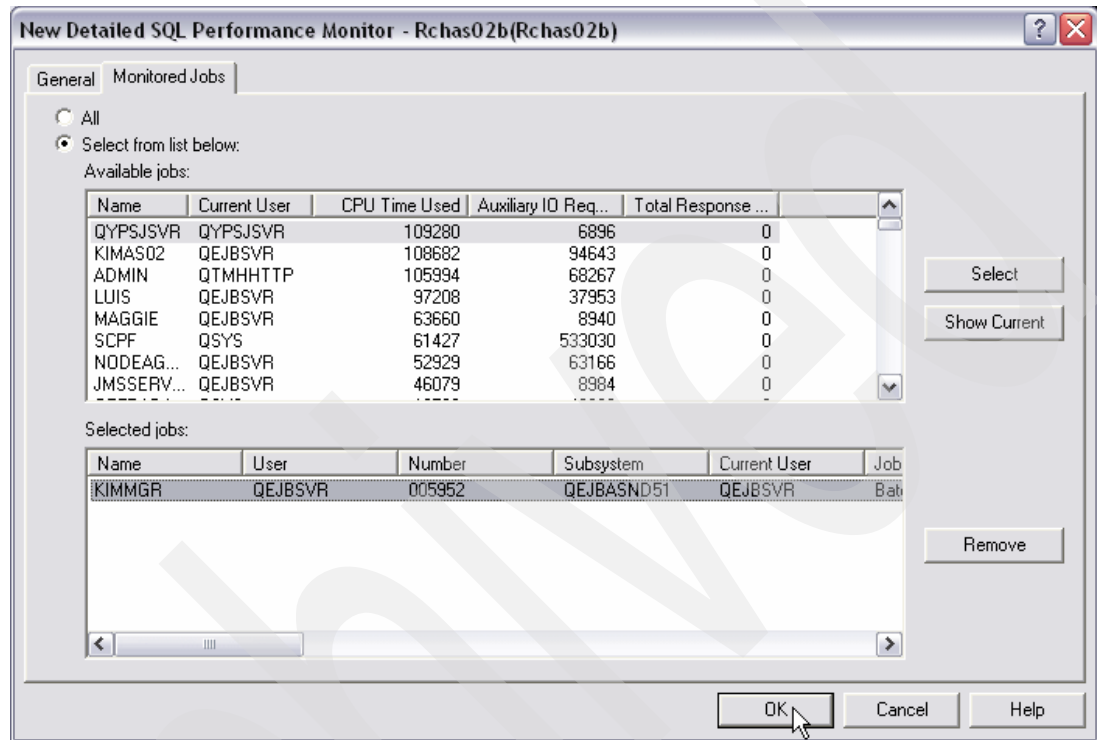


Figure 4-77 Selecting the job for database monitor

Working with existing SQL Performance Monitors

To start working with the existing SQL performance monitors, you click an SQL Performance Monitors container as shown in Figure 4-78.

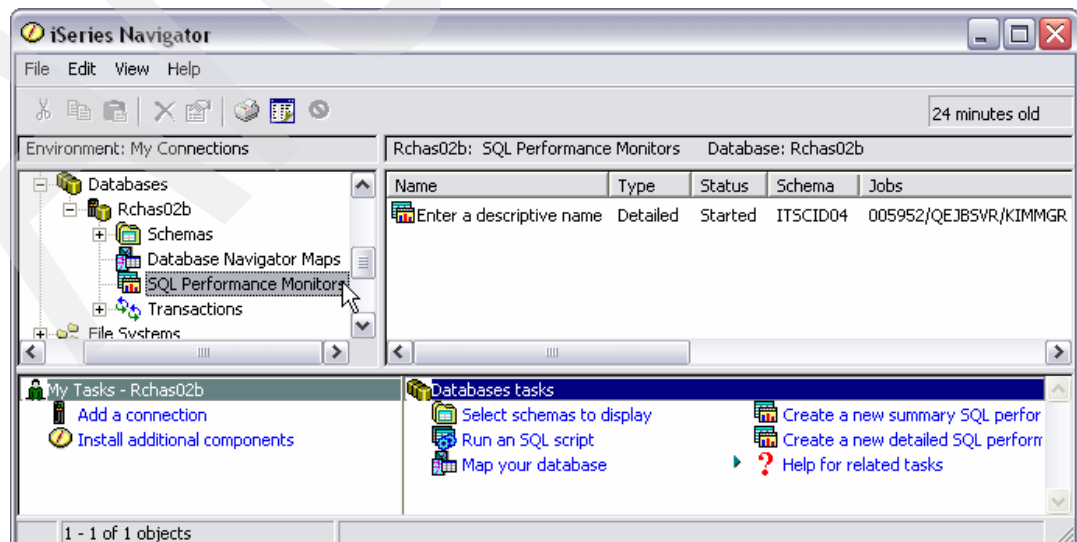


Figure 4-78 Selecting the Database Monitor to work with

You can either pause or end an active collection to analyze the results or choose to analyze the results while the collection is still running. Simply right-click a collection and choose the necessary option as shown in Figure 4-79. From this same menu, you can delete an obsolete database monitor collection by selecting the Delete option.

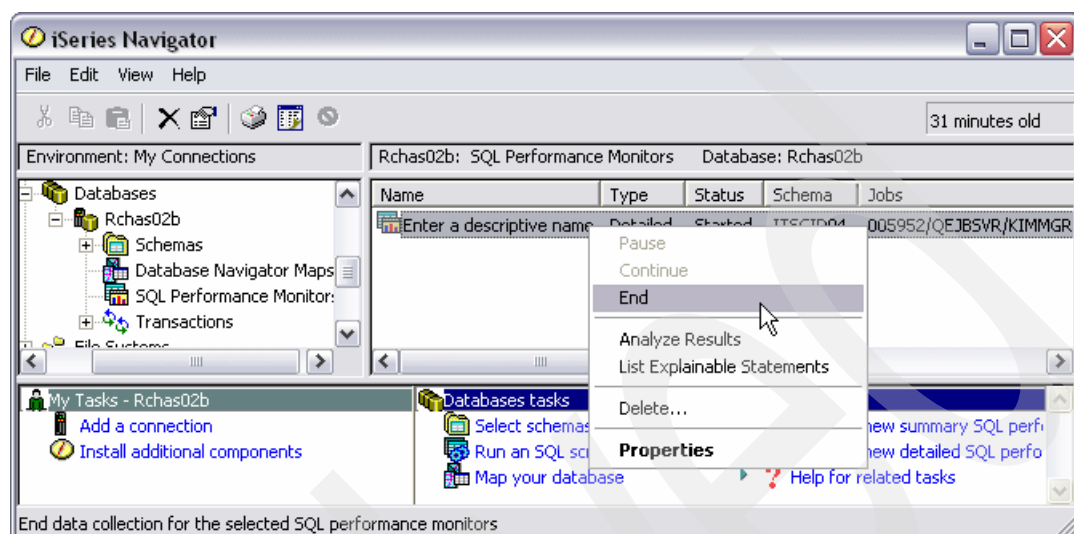


Figure 4-79 Ending a Database Monitor

4.2.9 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 use it to look for indexes that are “begging” to be created.

It is best to first concentrate on repetitious non-reusable Open Data Paths (ODP), 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, address those problems as well. Fine tune smaller problems after you address large problems. Generally, you use indexes to tune most performance problem areas.

4.2.10 SQL Visual Explain

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

You can use Visual Explain:

- ▶ On SQL statements that are stored in Database Performance Monitor data
- ▶ On detailed monitor data (STRDBMON or Detailed SQL Performance Monitor)
- ▶ Via the Recent SQL Performance Monitors task in iSeries Navigator’s SQL Script Center
- ▶ On SQL statements in the SQL Script Center

Monitor data has to be collected on a level of V4R5 or later of the operating system to guarantee that all of the information is available for drawing the picture. You may also collect data on one system and analyze it on another one.

Running Visual Explain proactively

Running Visual Explain via Run an SQL Script task is sometimes called *running Visual Explain proactively*, because you can:

- ▶ Run the query
- ▶ Make changes
- ▶ Run it again

This allows you to see how the changes affect the implementation. However, if your query is long running, you may want to collect 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 to be explained has already run. However, if your query is a long running one, 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 the Run SQL Script task (without actually running it):

1. In the iSeries Navigator window, expand **your server** → **Database** → **your database** → **SQL Performance Monitors**.
2. Select the detailed SQL performance monitor you want to use, right-click, and select **List Explainable Statements**.
3. In the SQL statements monitored list, select the SQL statement you want to visually explain. Click **Visual Explain**

The SQL statement is displayed as a graph in the Visual Explain window (Figure 4-80).

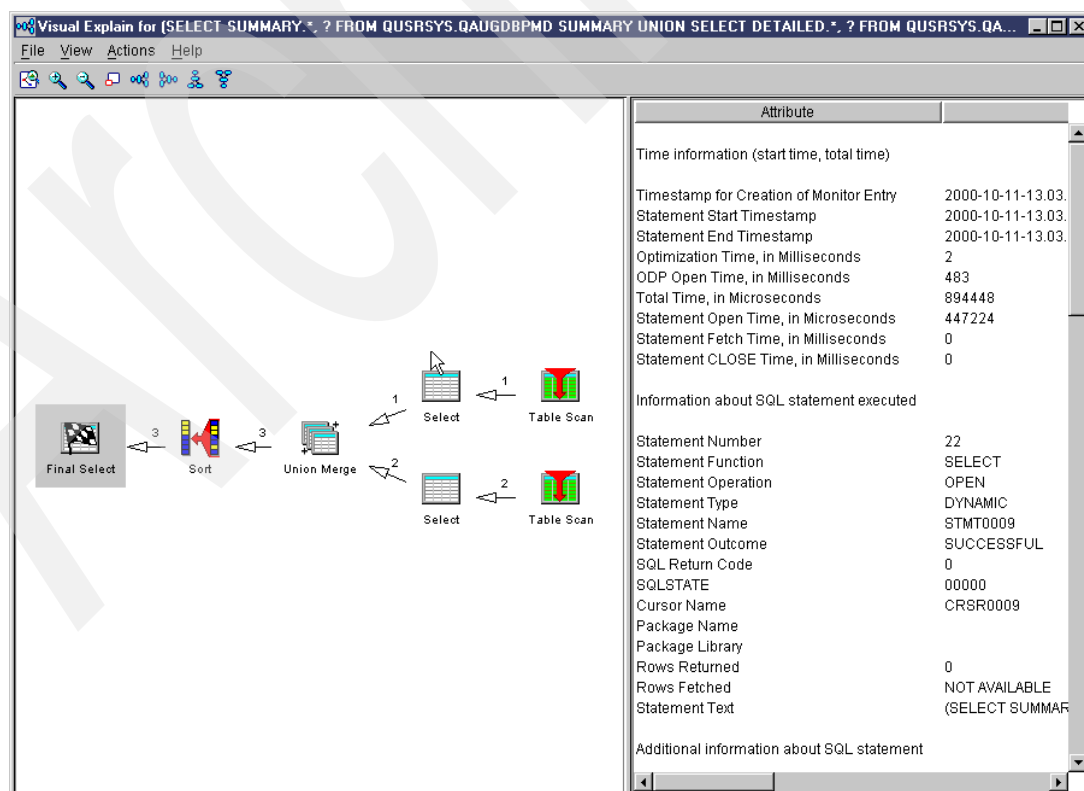


Figure 4-80 Example of Visual Explain

Executing Visual Explain from Run SQL Scripts

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

1. In the iSeries Navigator window, expand **your server** → **Database**. Right-click **Database** and select **Run SQL Scripts**.
2. If you have an existing SQL statement that you want to use, copy that statement and paste it into the Run SQL Scripts window. If you do not have an existing SQL statement, type a statement in the Run SQL Scripts window.
3. 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 can take a significant amount of time, but the information displayed is more complete and accurate.

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

4.3 System-level tools for reactive use

To use these set of tools, you follow these steps:

1. Use Collection Services to collect the data.
2. Create the data files from the collection object.
3. Use Performance Tools (LPP 5722-PT1) to determine where the bottleneck is hiding.

Performance Tools can generate the following type of data:

- ▶ Reports
- ▶ Display Performance Data (DSPPFRDTA) command
- ▶ Analyze Performance Data (ANZPFRDTA) command

4.3.1 Collecting performance data

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

- ▶ From a 5250 (green-screen) display
- ▶ iSeries 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 (Figure 4-81) appears. Choose option 2 (Collect performance data).

```
PERFORM                      IBM Performance Tools for AS/400
                               System:      SYSTEMA

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 4-81 Performance Tools for iSeries Main Menu

2. The Collect Performance Data display (Figure 4-82) appears. Choose option 1 (Start collecting data).

```
                        Collect Performance Data
                               SYSTEMA
                               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 4-82 Collect Performance Data display

3. The Start Collecting Data display (Figure 4-83) appears. Complete the parameters for the collection. To minimize the performance impact, we recommend that you do not create the database files when collecting the data (indicated by *NO in Figure 4-83). See the help text or *Performance Tools for iSeries*, SC41-5340, for a detailed explanation about the parameter values. The shorter the collection interval is that you specify, the more granular the collected the data is. Press Enter.

Start Collecting Data

Type choices, press Enter.

| | | |
|-------------------------------------|------------|--|
| Library | YOURLIB | Name |
| Collection interval (minutes) . . . | 5.00 | 0.25, 0.5, 1, 5, 15, 30, 60 |
| Retention period: | | |
| Days | 5 | *PERM, 0-30 |
| Hours | 0 | 0-23 |
| Cycling: | | |
| Time to synchronize cycle | 00:00:00 | HH:MM:SS |
| Frequency to cycle collections . . | 4 | 1-24 |
| Create database files | *NO | *YES, *NO |
| Collection profile | *STANDARDP | *MINIMUM, *STANDARD, *STANDARDP, *ENHPCPLN |

F3=Exit F12=Cancel

Figure 4-83 Start Collecting Data display

Collecting performance data with iSeries Navigator

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

1. Start iSeries Navigator.
2. Expand the iSeries server for which you want to start the data collection. Expand **Configuration and Service**. Select **Collection Services**, right-click, and select **Start Collecting** (see Figure 4-84).

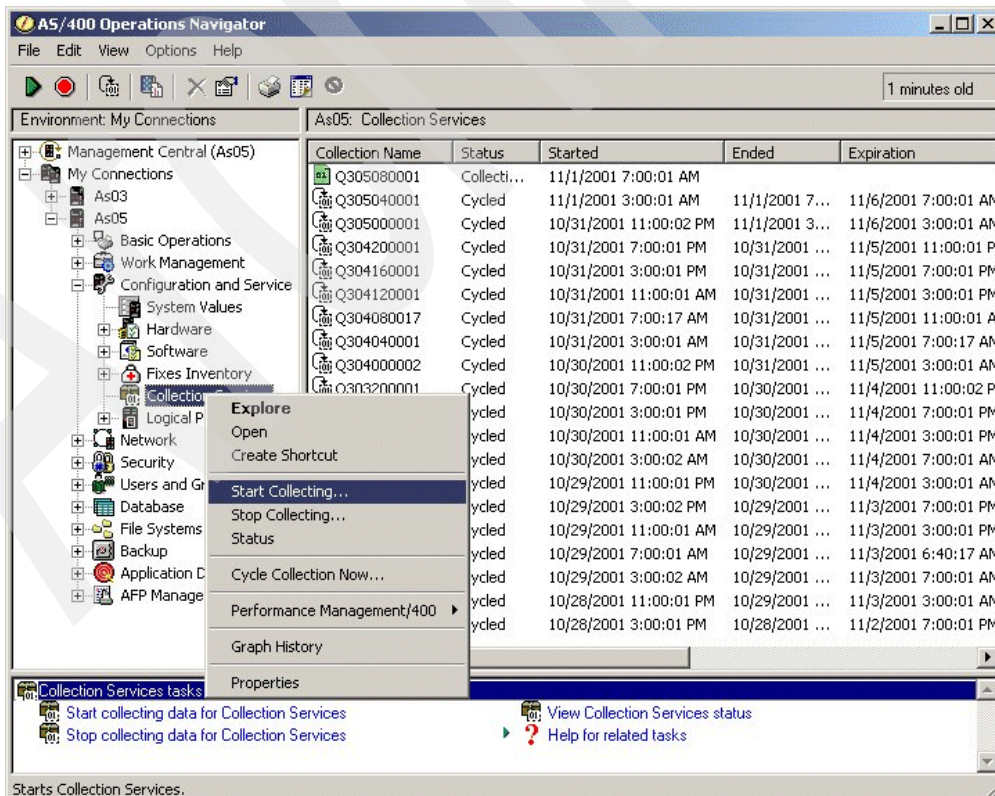


Figure 4-84 Starting Performance Data Collection with iSeries Navigator

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

4.3.2 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. However, because of the system overhead, we recommend that you collect the data and create the files only when you need them. Sending a collection object to another system is considerably lighter than sending a library.

Creating performance data using a 5250 display

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

1. Enter the Create Performance Data (CRTPFRTA) command. Press F4.
2. In the Create Performance Data display (Figure 4-85), specify the values for the parameters and press Enter to create the database files.

| Create Performance Data (CRTPFRTA) | | |
|------------------------------------|-------------|------------------------------|
| Type choices, press Enter. | | |
| From collection | _____ | Name, *ACTIVE |
| Library | QPFRTA | 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 4-85 Create Performance Data display

Creating performance data with iSeries Navigator

Perform the following steps:

1. Start iSeries Navigator.
2. As shown in Figure 4-86, expand the iSeries server from which you collected the data. Expand **Configuration and Service**. Double-click **Collection Services**.
3. In the right pane shown in Figure 4-86, right-click your data collection, and select **Create Database Files Now**.

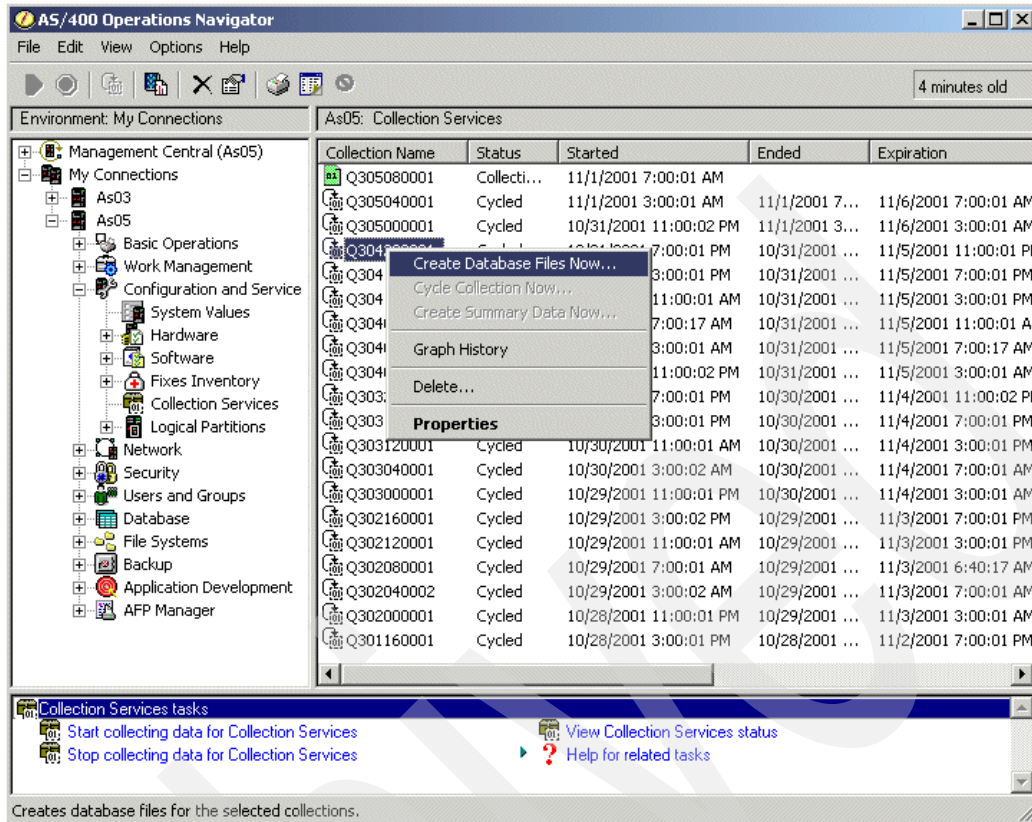


Figure 4-86 Creating the Performance Data with iSeries Navigator

- The Create Database Files from window (Figure 4-87) opens. Specify the library and the name for the database member and click **OK**.

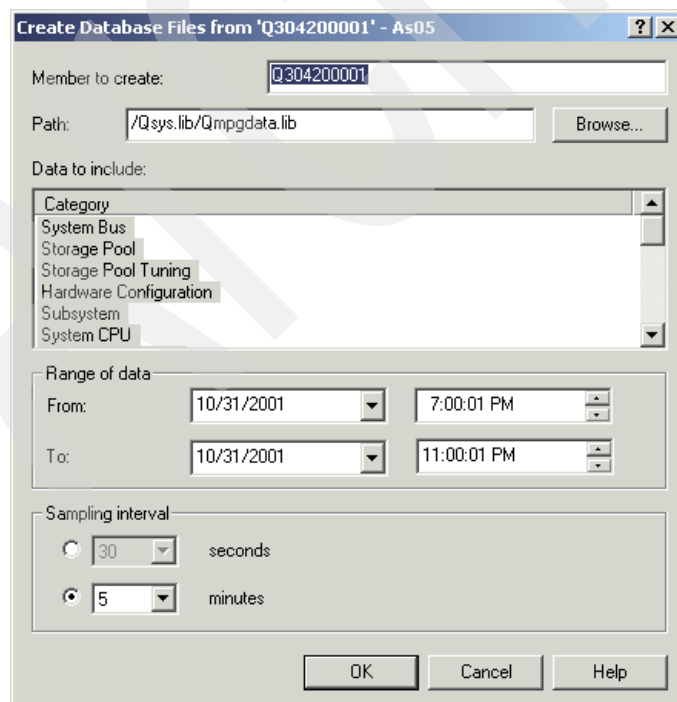


Figure 4-87 Create Database File window

4.3.3 Performance Tools for iSeries

The Performance Tools for iSeries LPP includes tools for collecting data about the performance on a system, job, or program level. It also includes tools to print reports and graphics from 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 CRTPFRTA command to create a set of database files from the performance information that is stored in a management collection (*MGTCOL) object. You can learn more about the database files in the iSeries Information Center, under **Systems Management** → **Performance**:

<http://publib.boulder.ibm.com/pubs/html/as400/v5r1/ic2924/index.htm>

To create the database files, see 4.3.2, “Creating performance data” on page 96.

DSPPFRDTA command

The 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 4-88 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 : | 10/25/66 | 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 4-88 DSPPFRDTA example

Reports

Based on the data collected on the system, Performance Tools for iSeries provides 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 collect 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. It 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. It may recommend changes to basic system tuning values and list conclusions about conditions that can 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 Advisor's conclusions to:

- ▶ Make changes to your system
- ▶ Guide further performance data collection
- ▶ Help you request performance reports that contain 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
- ▶ I/O processor (IOP) 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

4.3.4 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, you must first check 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:

- ▶ WRKACTJOB (See the examples in 4.2.3, "WRKACTJOB command" on page 31.)
- ▶ WRKSYSACT

Main storage

Use the WRKSYSSTS command to investigate faulting and the wait-to-ineligible transitions. See 4.2.1, “WRKSYSSTS command” on page 28, for details about using this command.

Disk

Use the WRKDSKSTS command to determine if you have enough disk space and disk units. This is discussed in 4.2.2, “WRKDSKSTS command” on page 29.

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: An A to Z Reference for Problem Determination*, SG24-8253, for details about 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 locks

Use 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 “Using PEX to collect performance data” on page 227.

4.3.5 Performance Management for iSeries, PM/400, and PM eServer iSeries

Performance Management/400 (now PM eServer iSeries, but previously known as PM/400) is an automated and self-managing tool. It uses Collection Services to gather the nonproprietary performance and capacity data from your server and then sends the data to IBM. When you send your data to IBM, you eliminate the need to store all the trending data yourself. IBM stores the data for you and provides you with a series of reports and graphs that show your server's growth and performance. You can access your reports electronically using a traditional browser.

The PM eServer iSeries service includes a set of reports, graphs, and profiles that help you maximize current application and hardware performance (by using performance trend analysis). These tools also help you to understand better (by using capacity planning) how your business trends relate to the timing of required hardware upgrades such as CPU or disk. Capacity planning information is provided by trending system utilization resources and throughput data, which can be thought of as an early warning system for your servers. You may think of PM eServer iSeries as a virtual resource that informs you about the health of your system.

PM eServer iSeries uses less than 1% of your CPU. It uses approximately 58 MB of disk space, which depends on your hardware model and the size of your collection intervals.

4.4 Java and application-level tools

Java and application-level tools refer to tools that are capable of looking “under the covers” of your running application. Several tools are available, each with various levels of complexity and different forms of output.

For a quick look at what is going on in your JVM, use the Dump Java Virtual Machine (DMPJVM) command. To compare two snapshots of your JVM over a specified interval, execute the Analyze Java Virtual Machine (ANZJVM) command. For more in-depth management and analysis, you may use either the HEAPANA command or the JAVAGCTOOLS in the SST interface.

4.4.1 DMPJVM command

The DMPJVM command is a standard operating system command that dumps information about the JVM for a specified job. The information that is produced is directed to a printer file QDMPJVM. The user data for the QDMPJVM file is DMPJVM. The dump includes formatted information about the classpath, garbage collection, and threads associated with the JVM as follows:

- ▶ The classpath
- ▶ Garbage collection
- ▶ Threads associated with the JVM

Figure 4-89 shows the classpath information output from the DMPJVM command. It also includes information about the version of the JDK that was used.

```
Mon Jul 12 17:23:49 2004
Java Virtual Machine Information 005325/QEJB5VR/SERVER1
.....
. Classpath
.....
java.version=1.4
sun.boot.class.path=/QIBM/ProdData/WebAS51/Base/java/ext/ibmorb.jar:/QIBM/ProdData/WebAS51/Base/java/ext/ibmext.jar:/QIBM/ProdData/WebAS51/Base/java/endorsed/xml.jar:/QIBM/ProdData/Java400/jdk14/lib/jdkptf14.zip:/QIBM/ProdData/CAP/jsse14.jar:/QIBM/ProdData/Java400/jdk14/lib/IBMiSeriesJSSE.jar:/QIBM/ProdData/Java400/jdk14/lib/jce.jar:/QIBM/ProdData/Java400/jdk14/lib/jaas.jar:/QIBM/ProdData/Java400/jdk14/lib/certpath.jar:/QIBM/ProdData/OS400/Java400/ext/ibmpkcs.jar:/QIBM/ProdData/OS400/Java400/ext/ibmjgssprovider.jar:/QIBM/ProdData/Java400/jdk14/lib/security.jar:/QIBM/ProdData/Java400/jdk14/lib/rt.jar:/QIBM/ProdData/Java400/jdk14/lib/charsets.jar:/QIBM/ProdData/Java400/jdk14/lib/sunrsasign.jar:/QIBM/ProdData/OS400/Java400/ext/IBMmisc.jar:/QIBM/ProdData/Java400/
java.class.path=/QIBM/UserData/WebAS51/Base/luis/properties:/QIBM/ProdData/WebAS51/Base/properties:/QIBM/ProdData/WebAS51/Base/lib/bootstrap.jar:/QIBM/ProdData/WebAS51/Base/lib/j2ee.jar:/QIBM/ProdData/WebAS51/Base/lib/lmproxy.jar:/QIBM/ProdData/WebAS51/Base/lib/urlprotocols.jar:/QIBM/ProdData/WebAS51/Base/deploytool/itp/plugins/org.eclipse.core.resources_2.1.1/resources.jar
java.ext.dirs=/QIBM/ProdData/WebAS51/Base/java/wsa400ext:/QIBM/ProdData/WebAS51/Base/java/ext:/QIBM/ProdData/Java400/jdk14/lib/ext:/QIBM/UserData/Java400/ext
java.library.path=/QSYS.LIB/QEJBAS51.LIB:/QSYS.LIB/QGPL.LIB:/QSYS.LIB/QTEMP.LIB:/QSYS.LIB/QDEVELOP.LIB:/QSYS.LIB/QBLDSYS.LIB:/QSYS.LIB/QBLDSYSR.LIB
```

Figure 4-89 DMPJVM output: Classpath detail

Figure 4-90 shows the garbage collection information produced by the DMPJVM command. Using this 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: 98304 K
  Max size: 240000000 K
Current values
  Heap size: 513920 K
  Garbage collections: 202
Additional values
  JIT heap size: 106720 K
  JVM heap size: 227408 K
  Last GC cycle time: 19606 ms

```

Figure 4-90 DMPJVM output: Garbage collection detail

Figure 4-91 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.

```

Thread: 00000036 ServerSocket[addr=0.0.0.0/0.0.0.0,port=0,localport=16013]
  TDE: B00110000D0D9000
  Thread priority: 5
  Thread status: Timed wait
  Thread group: main
  Runnable: com/ibm/ws/webcontainer/http/HttpTransport
Stack:
  java/net/PlainSocketImpl.accept(Ljava/net/SocketImpl;)V+5
(PlainSocketImpl.java:353)
  java/net/ServerSocket.implAccept(Ljava/net/Socket;)V+49 (ServerSocket.java:448)
  java/net/ServerSocket.accept()Ljava/net/Socket;+46 (ServerSocket.java:419)
  com/ibm/jsse/bg.accept()Ljava/net/Socket;+1 (:0)
  com/ibm/ws/http/HttpTransport.run()V+0 (HttpTransport.java:225)
  java/lang/Thread.run()V+11 (Thread.java:534)
Locks:
  None

```

Figure 4-91 DMPJVM output: Information for the first thread

The DMPJVM command is an excellent tool that provides a quick overview of your running application.

4.4.2 ANZJVM command

The ANZJVM command collects information about the JVM for a specific job. An initial set of JVM information is collected immediately when you run the command. This first set of collected JVM data is called a *snapshot*. Then, a second snapshot is taken at the time specified by the INTERVAL parameter of the ANZJVM command.

By taking an initial snapshot of the JVM and then comparing the data with a snapshot taken at a later time, the data can be compared and analyzed to help find object leaks and garbage collection information. The information is dumped to the printer file QANZJVM. The user data

for the QANZJVM file is ANZJVM. The dump includes formatted information about the JVM heap.

Restriction: The ANZJVM command uses the Start Service Job (STRSRVJOB) and Start Debug (STRDBG) commands. The user who runs the ANZJVM command must have authority to use these commands.

Figure 4-92 shows an example of running the ANZJVM command against job MY_JAVA_VM with forced garbage collection and an interval of 60 seconds for the second snapshot.

| Analyze Java Virtual Machine (ANZJVM) | |
|--|--------------|
| Type choices, press Enter. | |
| Job name JOB | > MY_JAVA_VM |
| User | > JAVA_USER |
| Number | > 004343 |
| Time interval INTERVAL | 60 |
| Force garbage collection FRCGC | *YES |
| Sort by SORT | *NUMOBJCHG |

Figure 4-92 ANZJVM: Executing ANZJVM to compare two JVM snapshots

To obtain a cleaner view of the heap, we recommend that you look at it as soon as possible after a garbage collection cycle. The ANZJVM command has an FRCGC parameter to specify if garbage collection should be forced. In general, you usually want to run the ANZJVM command with forced garbage collection.

If you examine the QANZJVM spooled file after you run the ANZJVM command, you will notice the following information for each class in the heap:

- ▶ Class name
- ▶ Garbage collection heap information
 - Pass one
 - Pass two
 - Change in the number of objects in the garbage collection heap
- ▶ Object space used
 - Pass one
 - Pass two
 - Change in object size
- ▶ Global registry information, which is the same for the global registry as what is listed for the object table
- ▶ Loader name

```
Tue Jul 13 10:28:40 2004
Job: 009281/QEJBSVR/SERVER1
Interval: 60 (SEC)
Total garbage collection cycles prior to running: 101
Total garbage collection cycles after running: 106
GC forced: YES
TIME OF FORCED GC: Tue Jul 13 10:26:57 2004
TIME OF FORCED GC: Tue Jul 13 10:28:23 2004
```

Apart from general information, the ANZJVM output includes classloader information as shown in Figure 4-94.

```
.....  
. Class loader information .  
.....  
0 Default class loader  
1 com/ibm/ws/classloader/CompoundClassLoader  
2 com/ibm/ws/webcontainer/jsp/servlet/JasperLoader  
3 sun/reflect/DelegatingClassLoader  
4 com/ibm/ws/bootstrap/ExtClassLoader  
5 sun/misc/Launcher$AppClassLoader  
6 sun/misc/Launcher$ExtClassLoader
```

The ANZJVM output also includes comparative garbage collection (GC) heap information from the two snapshots. An example of this output is shown in Figure 4-95.

```

.....
. GC heap information
.....
Loader
|
|   Number of pass one objects in the GC heap
|   Number of pass two objects in the GC heap
|   Change in the number of objects in the GC heap
|   Pass one object size (K)
|   Pass two object size (K)
|   Change in object size (K)
|   In global registry
|   Class name
|
4  61200      84000      22800      2868      3937      1069      NO   com/ibm/ws/webcontai
4  323102     341328     18226     15145     15999     854      NO   com/ibm/ws/webcontai
0  118782     133508     14726     5567      6258     691      NO   java/util/Hashtable$
4  161551     170659     9108      8203      8666     463      NO   com/ibm/ws/webcontai
0  164683     171559     6876     17190     18177     987      YES  [Ljava/lang/Object;
0  9945       15346     5401      1130      1266     136      NO   [I
0  3171       8363      5192      161       424      263      NO   java/math/BigInteger

```

104 Maximum Performance with WebSphere Application Server V5.1 on iSeries

Finally, as shown in Figure 4-96, statistics on the global registry are also included.

| Global registry information | | | | | | | |
|---|---|-----|---|-----|--|---------------------------|------------------------|
| Loader | | | | | | | |
| | Number of pass one objects in the GC heap | | Number of pass two objects in the GC heap | | Change in the number of objects in the GC heap | | |
| | | | | | Pass one object size (K) | Pass two object size (K) | |
| | | | | | | Change in object size (K) | Class name |
| 4 | 586 | 623 | 37 | 96 | 102 | 6 | com/ibm/ws/util/Thread |
| 4 | 1 | 1 | 0 | 0 | 0 | 0 | com/ibm/ws/messaging/B |
| 0 | 878 | 878 | 0 | 109 | 109 | 0 | java/lang/Thread |
| 4 | 2 | 2 | 0 | 0 | 0 | 0 | com/ibm/disthub/impl/u |
| 4 | 2 | 2 | 0 | 0 | 0 | 0 | com/ibm/disthub/impl/u |
| 4 | 1 | 1 | 0 | 0 | 0 | 0 | com/ibm/ws/Transaction |
| 0 | 285 | 285 | 0 | 12 | 12 | 0 | java/io/RandomAccessFi |

Figure 4-96 ANZJVM output: global registry information

Using the ANZJVM command to locate memory leaks

The ANZJVM command can help you find object leaks because it takes two copies of the garbage collection heap that are separated by a specified time interval. To locate object leaks, you can look at the number of instances of each class in the heap. You should note classes that have an unusually high number of instances as possibly leaking.

You may also note the change in number of instances of each class between the two copies of the garbage collection heap. If the number of instances of a class continually increases, that class should be noted as possibly leaking.

The longer the time interval between the two copies, the more certainty you have that objects are actually leaking. By running the ANZJVM command several times with a larger time interval, you should be able to easily diagnose what is leaking.

Important: Due to the length of time that the ANZJVM command can run, it is highly possible that a JVM may end before the command can complete. In the event that the JVM ends, the ANZJVM command returns the JVAB606 message (that JVM ended while processing the ANZJVM command) along with the data that it could obtain.

There is also no upper limit on the number of classes that a JVM can handle. If there are more classes than can be handled, the ANZJVM command should return the data that can be handled, along with a message letting you know that additional information was not reported. When the data requires truncation, the ANZJVM command returns as much information as possible.

The INTERVAL parameter is restricted to 3600 seconds (one hour) in length. The number of classes that the ANZJVM command can return information about is only limited by the amount of storage on your system.

4.4.3 JAVAGCTOOLS (STRSST)

On some occasions you may want to gather information for a running JVM when the system is in a degraded state. Rather than use the DMPJVM command, consider using the JAVAGCTOOLS macro that is available from within the SST interface.

1. To access JAVAGCTOOLS, enter the STRSST command as shown in Figure 4-97.

Restriction: To access SST, your profile must have *SERVICE authority. In addition, you must have a SST user ID and password. Your system administrator should be able to supply this information if you are authorized to use SST.

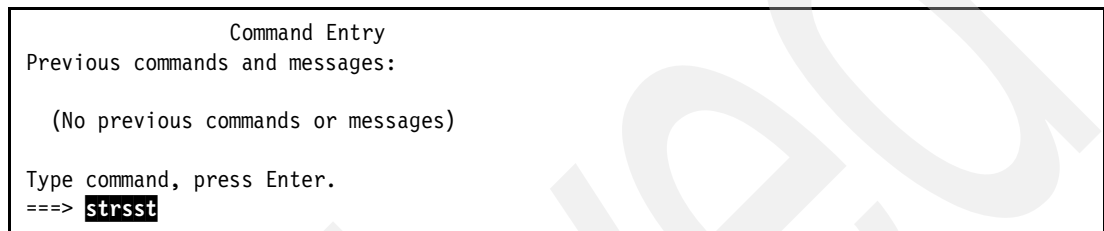


Figure 4-97 JAVAGCTOOLS: STRSST command

2. Enter your SST user ID and password
3. Now you see the main System Service Tools (SST) menu (Figure 4-98). Enter 1 (Start a service tool).

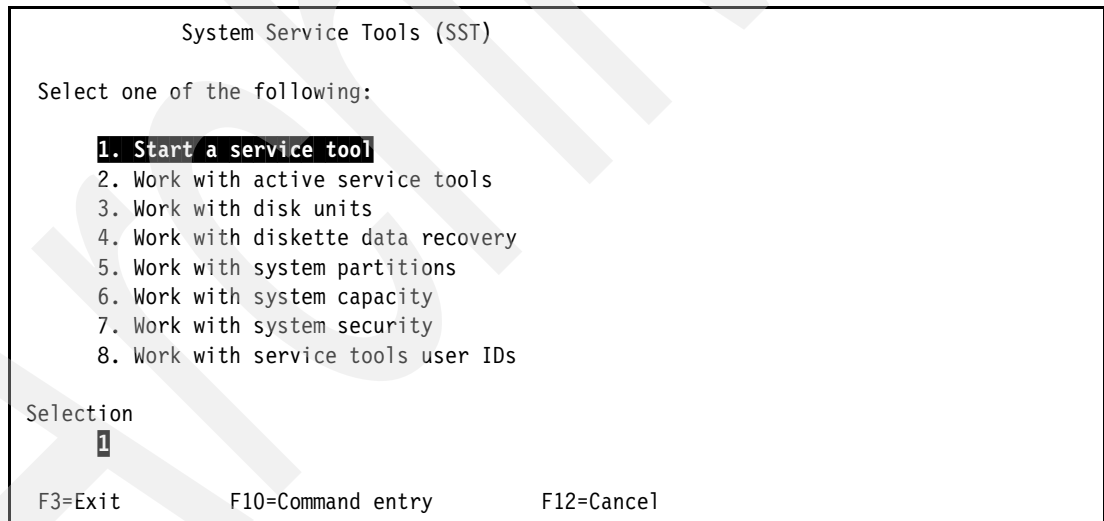


Figure 4-98 JAVAGCTOOLS: Start a service tool

4. Upon starting a service tool, you see a warning message (Figure 4-99) indicating that improper use of SST can damage your system. Therefore, follow the instructions presented here carefully.

Type option 4 (Display/Alter/Dump) to access the display, alter, and dump functions.

Start a Service Tool

Warning: Incorrect use of this service tool can cause damage to data in this system. Contact your service representative for assistance.

Select one of the following:

1. Product activity log

2. Trace Licensed Internal Code

3. Work with communications trace

4. Display/Alter/Dump

5. Licensed Internal Code log

6. Main storage dump manager

7. Hardware service manager

Selection

4

F3=Exit F12=Cancel F16=SST menu

Figure 4-99 JAVAGCTOOLS: Display/Alter/Dump

5. Select the output device. As shown in Figure 4-100, type option 2 to send output information from JAVAGCTOOLS to the printer spooled file.

Display/Alter/Dump Output Device

Select one of the following:

1. Display/Alter storage

2. Dump to printer

4. Dump to media

6. Print media dump file

7. Display dump status

Selection

2

F3=Exit F12=Cancel

Figure 4-100 JAVAGCTOOLS: Dump to printer

6. As shown in Figure 4-101, select the data that you want to examine. For JAVAGCTOOLS, select option 2 to access the appropriate data.

```

Select Data

Output device . . . . . : Printer

Select one of the following:

    1. Machine Interface (MI) object
    2. Licensed Internal Code (LIC) data
    3. LIC module
    4. Tasks/Processes
    5. Starting address

Selection
    2

F3=Exit  F12=Cancel
```

Figure 4-101 JAVAGCTOOLS: Select data

7. In the Select LIC Data display (Figure 4-102), press the Page Down key to view more options.

```

Select LIC Data

Output device . . . . . : Printer

Select one of the following:

    1. Node Address Communication Area (NACA)
    2. Machine Initialization Status Record (MISR)
    3. Source/Sink Active Device List (SSADL)
    4. Recovery list
    5. Database in use table
    6. Attached commit block table
    7. Alter Log
    8. Machine index
    9. LIC Link Map
    10. Heap space

Selection

F3=Exit  F12=Cancel

More...
```

Figure 4-102 JAVAGCTOOLS: Select LIC Data

8. Type option 14 (Advanced analysis) as illustrated in Figure 4-103.

```

                                Select LIC Data

Output device . . . . . :  Printer

Select one of the following:

    11. Main storage usage trace
    12. Transport manager traces
    13. Storage management functional trace
    14. Advanced analysis
    15. Journal list
    16. Journal work segment
    17. Database work segment

Selection
    14

F3=Exit  F12=Cancel

```

Figure 4-103 JAVAGCTOOLS: Advanced Analysis

9. Now you see a list of available Advanced Analysis commands, as shown in Figure 4-104. Press the Page Down key to see additional Advanced Analysis options.

```

                                Select Advanced Analysis Command

Output device . . . . . :  Printer

Type options, press Enter.
  1=Select

Option  Command
      ADDRESSINFO
      BATTERYINFO
      CLUSTERINFO
      CONDITIONINFO
      COUNTERINFO
      DSTINFO
      EXCEPTCHAIN
      FINDFRAMES
      FINDPTF
      FINDRUNAME
      FINDSLICTOKEN
      FIXUPSIZE

F3=Exit  F12=Cancel
More...

```

Figure 4-104 JAVAGCTOOLS: Select Advanced Analysis Command display

10. You see an additional Advanced Analysis options display (Figure 4-105). Type option 1 (JAVAGCTOOLS).

```

                                Select Advanced Analysis Command

Output device . . . . . :  Printer

Type options, press Enter.
  1=Select

Option   Command

          FLIGHTLOG
          FRCA
          GATEINFO
          HANDLEMAPPERINFO
          IPCONFIG
          JAVAJITINFO
  1  JAVAGCTOOLS
          JVALOCKINFO
          LICLOG
          LLHISTORYLOG
          LOCKINFO
          MASOCONTROLINFO

F3=Exit  F12=Cancel

```

Figure 4-105 JAVAGCTOOLS: Selecting the JAVAGCTOOLS macro

11. In the Specify Advanced Analysis Options display (Figure 4-106), you see many different options that you can specify when running JAVAGCTOOLS. You may want to begin by exploring the available options. To do this, press F4.

```

                                Specify Advanced Analysis Options

Output device . . . . . :  Printer

Type options, press Enter.

Command . . . . . :  JAVAGCTOOLS

Options . . . . .

F3=Exit  F4=Prompt  F12=Cancel

```

Figure 4-106 JAVAGCTOOLS: Specify Advanced Analysis Options

For more detailed information about available options for JAVAGCTOOLS, refer to Table 4-3.

Table 4-3 JAVAGCTOOLS options

| Parameter(s) | Description |
|--|--|
| -h | Displays the help text for JAVAGCTOOLS. |
| -dumpJVMs | Searches for JVMs and dumps their addresses. This is the same as running JAVAGCTOOLS with no parameters. |
| -dumpJVM <i>VMaddress</i> | Dumps general information for the JVM address specified in the <i>VMaddress</i> variable. |
| -classTotals <i>VMaddress</i> | Dumps the active object totals for each class in the <i>VMaddress</i> variable. |
| -classTotals <i>VMaddress classAddr</i> | Dumps the active object totals for only the class specified in the <i>classAddr</i> variable. Note that the <i>VMaddress</i> variable must also be given. |
| -classDetails <i>VMaddress</i> | Dumps the active object details for each class within the <i>VMaddress</i> variable given. |
| -classDetails <i>VMaddress classAddr</i> | Dumps the active object details for only the class specified in the <i>classAddr</i> variable. Note that the <i>VMaddress</i> variable must also be given. |
| -objects <i>VMaddress</i> | Dumps the details for each active object and class for the value specified in the <i>VMaddress</i> variable. |
| -objects <i>VMaddress classAddr</i> | Dumps the details of each active object for only the class specified in the <i>classAddr</i> variable. Note that the <i>VMaddress</i> variable must also be supplied. |
| -object <i>VMaddress address</i> | Dumps the details of the Java object that is specified in the <i>address</i> variable. The <i>VMaddress</i> variable is also required. |
| -global <i>VMaddress</i> | Dumps the global registry for the JVM address specified in the <i>VMaddress</i> variable. |
| -threads <i>VMaddress</i> | For each thread in the JVM address given in the <i>VMaddress</i> variable, dumps the objects rooted in its stack. |
| -flight <i>VMaddress</i> | For each thread in the JVM address given in the <i>VMaddress</i> variable, dumps the objects in its flight recorder. |
| -refs <i>VMaddress</i> | Dumps the soft, weak, final, phantom, and global weak reference list for the JVM address supplied in the <i>VMaddress</i> variable. |
| -anchors <i>VMaddress</i> | Dumps the static fields of the classes found in the JVM address specified in the <i>VMaddress</i> variable. |
| -finalization <i>VMaddress</i> | Lists the finalization list for the JVM address specified in the <i>VMaddress</i> variable. |
| -leak <i>VMaddress</i> | Executes an object leak analysis with a default interval of 10 seconds for the JVM address specified in the <i>VMaddress</i> variable. |
| -leak <i>VMaddress seconds</i> | Performs an object leak analysis with an interval that is specified in the <i>seconds</i> variable. The JVM address must also be specified in the <i>VMaddress</i> variable. |
| -gc <i>VMaddress</i> | Executes a garbage collection cycle for the JVM specified in the <i>VMaddress</i> variable. |
| -scan <i>VMaddress address</i> | Dumps the address locations and objects for the <i>address</i> and <i>VMaddress</i> variables specified. |
| -tree <i>VMaddress address</i> | Dumps the object inheritance tree for the object specified in the <i>address</i> variable within the JVM specified in the <i>VMaddress</i> variable. |

| Parameter(s) | Description |
|--|---|
| -space <i>VMaddress</i> | Provides the blue, red, purple, object space used and object space free for every segment in the heap for the JVM specified in the <i>VMaddress</i> variable. |
| -verifySpace <i>VMaddress</i> | Dumps the same information provided in the -space parameter (above), but also shows whether vectors are in use. The JVM address must be supplied in the <i>VMaddress</i> variable. |
| -gcCycles <i>VMaddress</i> | Lists the garbage collection table for the last 500 garbage collection cycles for the JVM address provided in the <i>VMaddress</i> variable. |
| -nodes <i>VMaddress</i> | Shows the reuse segments in each node and counts the space in the segments. You will need to supplied the JVM address in the <i>VMaddress</i> variable. |
| -anzjvm <i>VMaddress</i> | Analyzes all of the JVMs referred to in the <i>VMaddress</i> variable, except for the object tree. The time-out for analysis and reporting is 5 seconds. |
| -anzjvmall <i>VMaddress</i> | Analyzes all of the JVMs referred to in the <i>VMaddress</i> variable, including the object tree. The time-out for analysis and reporting is 5 seconds. |
| -anzjvm <i>VMaddress aTime rTime</i> | Analyzes all of the JVMs referred to in the <i>VMaddress</i> variable, except for the object tree. The user can also specify the analysis time in seconds using the <i>aTime</i> variable as well as the reporting time using the <i>rTime</i> variable. |
| -anzjvmall <i>VMaddress aTime rTime</i> | Analyzes all of the JVMs referred to in the <i>VMaddress</i> variable, including the object tree. The user can also specify the analysis time in seconds using the <i>aTime</i> variable as well as the reporting time using the <i>rTime</i> variable. |
| -anzjvmot1 <i>VMaddress objToExamine</i> | Executes an analysis of the JVM address provided in the <i>VMaddress</i> variable, including only the object tree starting at the <i>objToExamine</i> variable. The time-out value for analysis and reporting defaults to 5 seconds. |
| -anzjvmot1 <i>VMaddress objToExamine aTime rTime</i> | Executes an analysis of the JVM address provided in the <i>VMaddress</i> variable including only the object tree starting at the <i>objToExamine</i> variable. The time-out value for analysis is set based upon the <i>aTime</i> variable while reporting can be controlled via the <i>rTime</i> variable. |
| -anzjvmot2 <i>VMaddress classToExamine</i> | Performs an analysis of the JVM address given in the <i>VMaddress</i> variable for the object tree beginning at random <i>classToExamine</i> variable. The time-out defaults to 5 seconds for analysis and reporting. |
| -anzjvmot2 <i>VMaddress classToExamine aTime rTime</i> | Performs an analysis of the JVM address given in the <i>VMaddress</i> variable for the object tree beginning at random <i>classToExamine</i> variable. The time-out value for analysis is set based upon the <i>aTime</i> variable while reporting can be controlled via the <i>rTime</i> variable. |
| -anzjvmot3 <i>VMaddress classToExamine index</i> | Performs an analysis of the JVM address given in the <i>VMaddress</i> variable for the object tree beginning at indexed <i>classToExamine</i> variable. The time-out defaults to 5 seconds for analysis and reporting. |
| -anzjvmot3 <i>VMaddress classToExamine index aTime rTime</i> | Performs an analysis of the JVM address given in the <i>VMaddress</i> variable for the object tree beginning at indexed <i>classToExamine</i> variable. The time-out value for analysis is set based upon the <i>aTime</i> variable, while reporting can be controlled via the <i>rTime</i> variable. |

12. With JAVAGCTOOLS, locate the address of the JVM or JVMs that you want to examine. After looking at the available options using the F4 prompt (as in Figure 4-106), press F12.
13. You return to the Specify Advanced Analysis Options display (Figure 4-107). To output the addresses of available JVM or JVMs, press Enter and submit the JAVAGCTOOLS macro. Alternatively, you can type the option -dumpJVMs and then press Enter.

```

                                Specify Advanced Analysis Options

Output device . . . . . :   Printer

Type options, press Enter.

  Command . . . . . :   JAVAGCTOOLS

  Options . . . . . :   -dumpJVMs

F3=Exit   F4=Prompt   F12=Cancel

```

Figure 4-107 JAVAGCTOOLS: Obtaining the addresses of the available JVM or JVMs

14. In the Specify Dump Title display (Figure 4-108), enter meaningful text for the dump title. Press Enter.

```

                                Specify Dump Title

Output device . . . . . :   Printer

Type choices, press Enter.

  Dump title . . . . . :   DUMP JVM(S)

  Perform sizes . . . . . :   1           1=Yes, 2=No

  Partial print page numbers:
    From page . . . . . :           1       1-2147483647
    Through page . . . . . :   2147483647   1-2147483647

F3=Exit   F12=Cancel

```

Figure 4-108 JAVAGCTOOLS: Specifying a dump title

15. You see a message at the bottom of the display, indicating that the dump was submitted. Press F3 to exit.

Or you may see a message at the bottom of the display indicating that the dump completed, as shown in Figure 4-109. If you do not see the completion message, type option 7 to view the status of the dump. You can then refresh the status display until the dump completes and you receive a completion message on the bottom of the display.

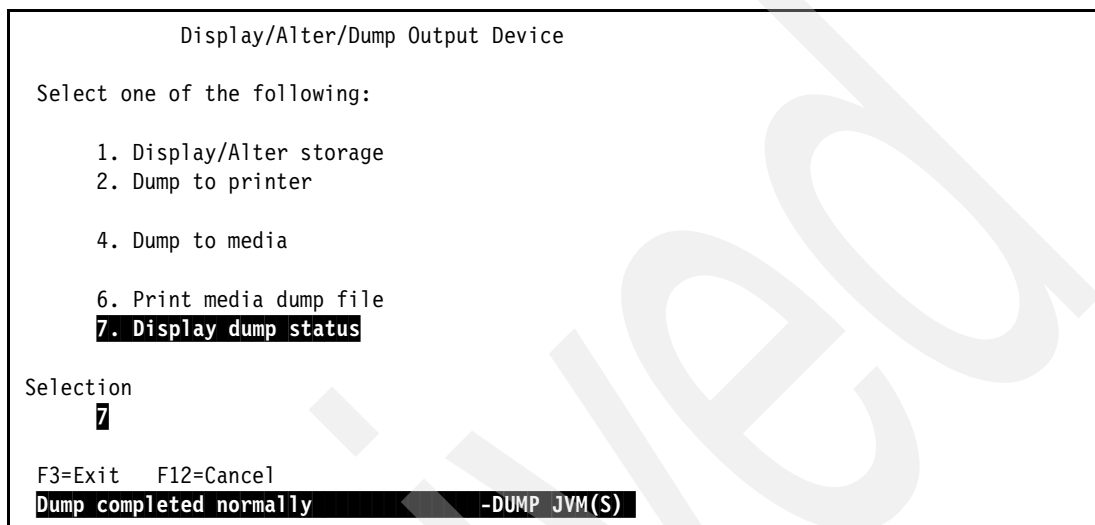


Figure 4-109 JAVAGCTOOLS: Receiving the dump completion status message

16. When you receive the completion message, you can exit SST if you choose, or press F3 and then F10 to access the command entry screen.

17. View the spooled output for JAVAGCTOOLS. The output is contained in the QPCSMPT file as shown in Figure 4-110. Type option 5 to examine the spooled file.

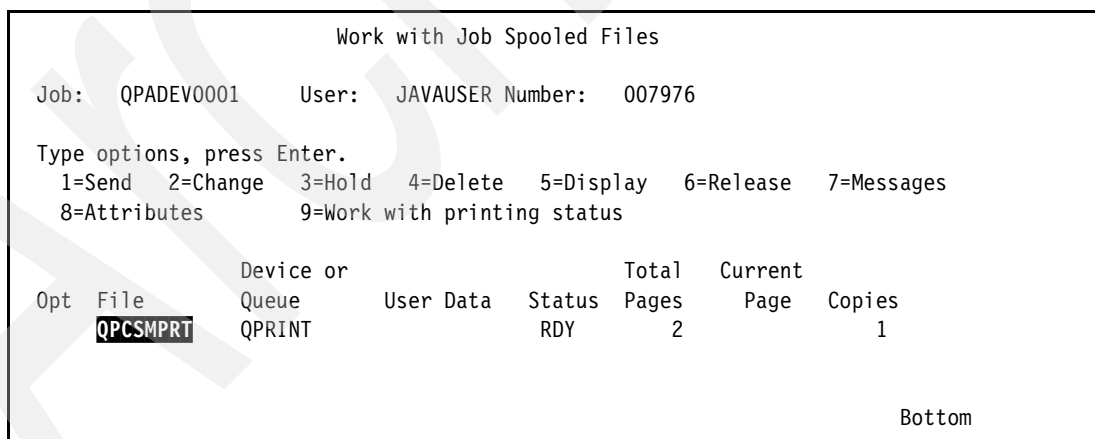


Figure 4-110 JAVAGCTOOLS: Viewing the output from JAVAGCTOOLS

18. You can look at the details for each JVM and note specifically its address, which takes the form VM*='VMaddress'. For example, in Figure 4-111, our JVM address is VM*=ECAD88ABF2008200.

```

DISPLAY/ALTER/DUMP                                DUMP JVM(S)
07/14/04 16:42:32
RUNNING MACRO: JAVAGCTOOLS -DUMPJVMs
USE ONE OF THE FOLLOWING VM* ADDRESSES AND RE-ISSUE THE MACRO.
JVM GENERAL INFORMATION:
VM*=ECAD88ABF2008200 TASK=B004C000AA4C000 TASK NAME=MITHREAD-ANAKIN QEJBSVR
007954
GC OBJECT*=EFABD616FE000A80 JAVA NEW*=DC93B94E00030680 CLEAR COLOR=5 MARK COLOR=7
ALLOCATION COLOR=7
GLOBAL REGISTRY ENTRIES=EFABD616FE004C70 EXTENTION=D018080DF9000020 JDK
VERSION=1_4PLU
NUMBER OF GC HELPER THREADS: TOTAL = 3 NODE 0 = 3 NODE 1 = 0 NODE 2 = 0 NODE 3 = 0
JVM HEAP SIZE INFORMATION. SIZES ARE IN BYTES:
INITIAL HEAP SIZE=1073741824 MAXIMUM HEAP SIZE=*NOMAX JIT HEAP=CB2AF3E078000048
SIZE=76185600
GC HEAP MANAGER*=C81DF144B3002000 SIZE=1369407488 JVM HEAP*=C81DF144B3000048
SIZE=215797760
INFORMATION. DURATION TIMES IN MILLISECONDS:
GARBAGE COLLECTION CYCLE INFORMATION. DURATION TIMES IN MILLISECONDS:
NUMBER GC CYCLES=2 CYCLE DURATION=1706 EMPTY OLD MARK SET DURATION=0 SYNC1
DURATION=871 SYNC1->SYNC2 DURATION=0
SYNC2 DURATION=0 SYNC2->ASYNC DURATION = 0 ASYNC HANDSHAKE DURATION=5 ASYNC->FINISH
TRACE DURATION=0
MARK GLOBALS DURATION=10 FINISH TRACE DURATION=426 REFERENCE OBJECT DURATION=30
SWEEP DURATION = 374
HEAP SIZE START(BYTES)=1252720640 HEAP SIZE END(BYTES)=956792832 START
TIMESTAMP=07/14/2004 16:29:46.046
FREE MEMORY SIZE START(BYTES)=135966553 FREE MEMORY SIZE END(BYTES)=897425097
REASON FOR COLLECTION: ALLOCATION THRESHOLD REACHED.

```

Figure 4-111 JAVAGCTOOLS: Obtaining the JVM address

19. With the JVM address, begin to explore other aspects of your running application. Re-enter SST as we did, and return to the Specify Advanced Analysis Options display for JAVAGCTOOLS.
20. Enter the options as shown in Table 4-3 on page 111 that correlate to the information you want to extract. For example, if you want to view the active object totals for each class, enter the `-classtotal`s option, followed by the address of your JVM in the JAVAGCTOOLS options display, as shown in Figure 4-112.

```

                                Specify Advanced Analysis Options

Output device . . . . . : Printer

Type options, press Enter.

Command . . . . . : JAVAGCTOOLS

Options . . . . . : -classtotals ECAD88ABF2008200

F3=Exit F4=Prompt F12=Cancel

```

Figure 4-112 JAVAGCTOOLS: Obtaining the active object totals for each class

21. Enter a dump title and wait for the dump completion message.
 22. Return to your spooled output where you see the spooled file associated with the class total dump. Select the option to display the class total dump.
- You see output similar to what is found in Figure 4-113.

```

DISPLAY/ALTER/DUMP07/14/04RUNNINGMACRO: JAVAGCTOOLS-CLASSTOTALSECAD88ABF2008200
***** BEGIN DUMP OF THE ACTIVE OBJECT TOTALS FOR EACH CLASS *****
JVM GENERAL INFORMATION:
VM*=ECAD88ABF2008200 TASK=B004C0000BB28000 TASKNAME=MITHREAD-ANAKINQEJBSVR 007954
GC OBJECT*=EFABD616FE000A80 JAVA NEW*=DC93B94E00030680 CLEAR COLOR=7 MARK COLOR=5
ALLOCATION COLOR=5 GLOBAL REGISTRY ENTRIES=EFABD616FE004C70 EXTENTION=D018080DF9000020
JDK VERSION=1_4PLU NUMBER OF GC HELPER THREADS: TOTAL = 3 NODE 0 = 3 NODE 1 = 0 NODE
2 = 0 NODE 3 = 0

JVM HEAP SIZE INFORMATION. SIZES ARE IN BYTES:
INITIAL HEAP SIZE=1073741824 MAXIMUM HEAP SIZE=*NOMAX JIT HEAP=CB2AF3E078000048
SIZE=77545472
GC HEAP MANAGER*=C81DF144B3002000 SIZE=1700626432 JVM HEAP*=C81DF144B3000048
SIZE=216322048

INFORMATION. DURATION TIMES IN MILLISECONDS:
GARBAGE COLLECTION CYCLE INFORMATION. DURATION TIMES IN MILLISECONDS:
NUMBER GC CYCLES=3 CYCLE DURATION=4101 EMPTY OLD MARK SET DURATION=0 SYNC1
DURATION=1606 SYNC1->SYNC2 DURATION=0
SYNC2 DURATION=459 SYNC2->ASYNC DURATION = 0 ASYNC HANDSHAKE DURATION=5
ASYNC->FINISH TRACE DURATION=0
MARK GLOBALS DURATION=10 FINISH TRACE DURATION=393 REFERENCE OBJECT DURATION=29
SWEEP DURATION = 1609
HEAP SIZE START(BYTES)=2279178240 HEAP SIZE END(BYTES)=1050378240 START
TIMESTAMP=07/14/2004 17:07:56.153
FREE MEMORY SIZE START(BYTES)=1147612898 FREE MEMORY SIZE END(BYTES)=991303231
REASON FOR COLLECTION: ALLOCATION THRESHOLD REACHED.
NUMBER TOTAL TOTAL
CLASS ADDRESS PROCESSED BLOCK SIZE OBJECT SIZE CLASS NAME
-----
F5714E32750033D0 156830 399139744 248571534 C
F5714E3275022820 11058 1415424 1150032 JAVA/LANG/CLASS
F5714E32752534B0 823 79008 72424 JAVA/LANG/PACKAGE
F5714E3275010FD0 328105 108754784 65642638 JAVA/LANG/STRING

```

Figure 4-113 JAVAGCTOOLS: Output of the active object totals for each class

Using the variety of options available with JAVAGCTOOLS, you can closely examine your JVM runtime if you are experiencing a performance problem.

4.5 WebSphere performance tools

WebSphere-level performance tools refer to tools that are capable of measuring the internal response time and resources used by the WebSphere Application Server. Measuring the performance of your application using the WebSphere Application Server runtime should commence as early in the development life cycle as possible. During developer unit testing and user acceptance testing, include several rounds of profiling using the facilities provided by the Profiling and Logging perspective of the WebSphere Studio family of development tools.

In addition, take advantage of the data surfaced by the Performance Monitoring Infrastructure. Access it using the included Tivoli Performance Viewer to closely monitor your application running on the WebSphere Application Server during the scalability testing phase of software development. Moreover, proactively analyze the information provided by the included Performance Advisors and the Log Analyzer. Make incremental changes over time so that you can maximize the power of the WebSphere Application Server.

4.5.1 Profiling applications running on the WebSphere Application Server

Profiling is the task of examining the runtime performance of an application. This information is critical for resolving performance-related problems within your application. If you do not have access to appropriate profiling information, you are most likely to spend an unnecessary amount of time trying to determine the source of the performance problem.

When dealing with application performance, try to optimize:

- ▶ *Execution time*, where the execution time is spent. This helps to identify time consuming methods.
- ▶ *Memory utilization*: Determine which objects are filling up the memory.

When you profile an application, keep in mind that the behavior of the application code can change based on the available data. If your production system has a table with 1 million records, but your development database only has 100 records, it may result in different runtime characteristics.

For example, most computers today have sufficient memory and processing power to easily deal with 100 records. The result may be that you do not see problems with caching or a performance problem in an algorithm, which happens only if you have more than 100000 records.

Also keep in mind the amount of interaction with the application when profiling. For a Web application, this is related to the number and diversity of requests. If you only make a single request to a component, you may end up with misleading data. One example of this is when using singletons and various pools, where initialization is expensive but subsequent requests are cheaper. If you record only one request, you will only see the expensive initialization.

Profiling architecture

The profiling facilities provided with the WebSphere Studio family of development tools, including WebSphere Development Studio Client for iSeries Advanced Edition, are designed to work in a client/server architecture. Figure 4-114 illustrates the profiling architecture.

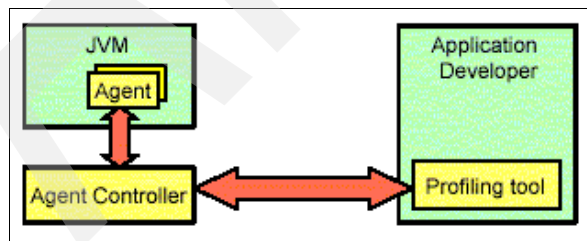


Figure 4-114 Profiling: Architecture

On the client-side, the profiling tool runs inside the integrated development environment (IDE). The profiling tool provides the developer with various views into the application. The Agent Controller runs on the server-side. It manages one or more agents running within the JVM or JVMs on the server.

During development, you may use the profiling tool within the test environment on your local development machine. Then, during user acceptance or scalability testing, you may profile your application once again on the iSeries server where the application is deployed.

The profiling capabilities include support for both local and remote profiling. With the profiling tools installed on the client-side, you merely need to make sure that the Agent Controller is available on the server.

Tip: Profiling is ideal while prototyping, during development, and throughout the remainder of the development life cycle prior to reaching production. Your system administrator may not recommend profiling in production unless absolutely necessary. We agree with this recommendation.

The Agent Controller is available on the distribution media that comes with your WebSphere development environment, for example, WebSphere Development Studio Client.

Profiling applications

The following sections examine more closely how to profile an application.

Starting the server for profiling

Before you can collect profiling information for your application, make sure that the Application Server is started in profiling mode. Assuming the Agent Controller is installed on the server, follow these steps:

1. Access the **server perspective** in your development environment.
2. Select the server where you want to start a profiling session, right-click, and select **Profile**.

This launches the server with *profiling* enabled and automatically opens the wizard to connect to the Agent Controller.

Profiling wizard

After successfully starting the server, the Profiling wizard opens.

1. In the Agent panel (Figure 4-115), select the agent. The available agents are:
 - **Java Profiling Agent:** This is a standard Java profiler. It is deployed with the agent controller and collects information about the execution of a Java program.
 - **J2EE Request Profiler:** This is made available by the Application Server. It has knowledge of J2EE and allows profiling information to be collected intelligently across containers and application servers in the Application Server.

Click **Next**.

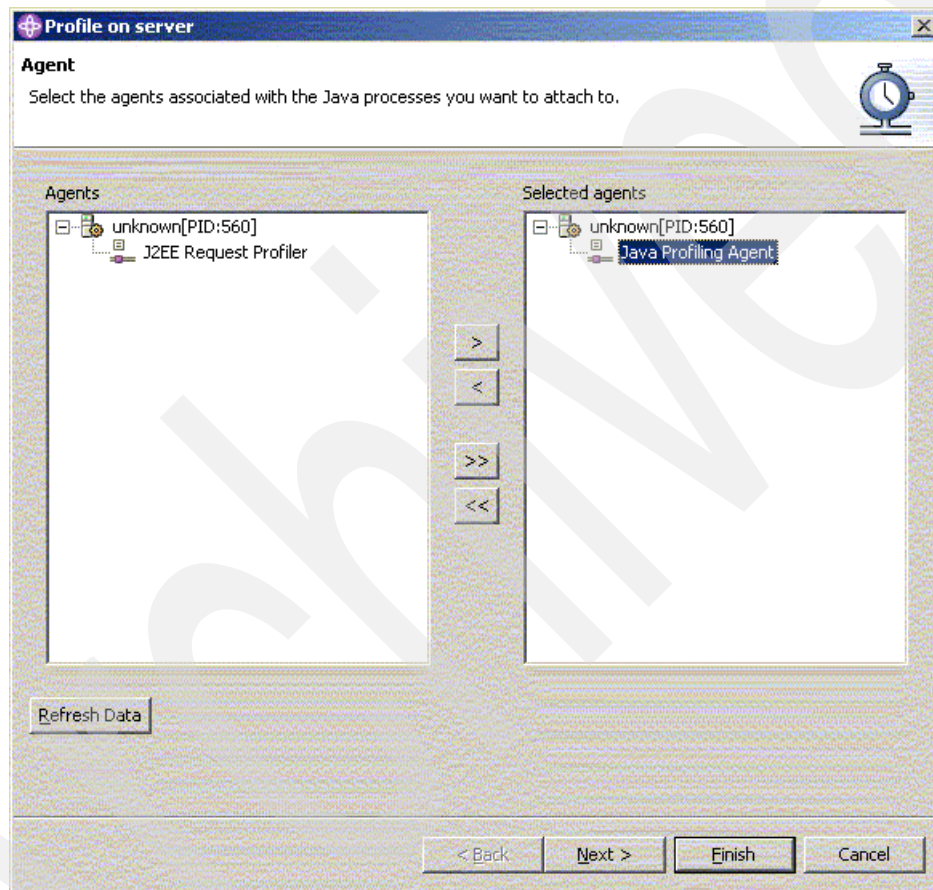
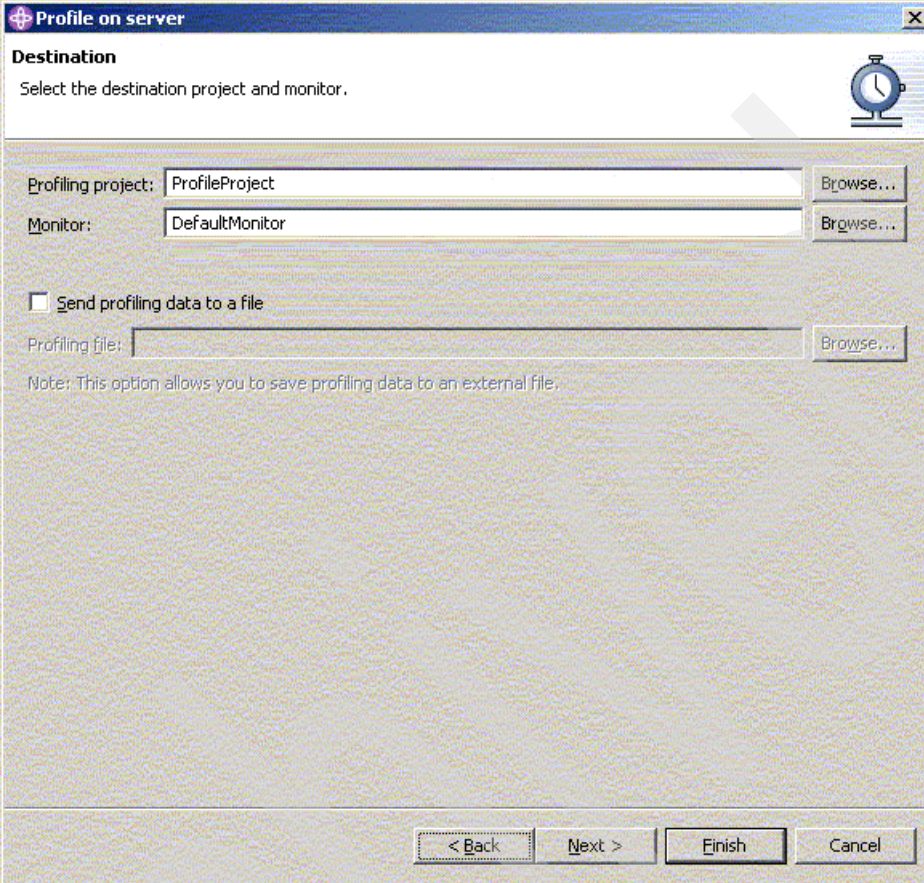


Figure 4-115 Profiling: Selecting the agent

2. In the Destination panel (Figure 4-116), specify the destination for the collected profiling data. Click **Next**.



The screenshot shows a Windows-style dialog box titled "Profile on server". The "Destination" tab is selected, indicated by a clock icon in the top right corner. The instruction "Select the destination project and monitor." is displayed. Below this, there are two text input fields: "Profiling project:" containing "ProfileProject" and "Monitor:" containing "DefaultMonitor". Each field has a "Browse..." button to its right. Below these fields is a checkbox labeled "Send profiling data to a file", which is currently unchecked. To the right of the checkbox is another "Browse..." button. Below the checkbox is a "Profiling file:" text input field, also with a "Browse..." button to its right. A note at the bottom of the main area states: "Note: This option allows you to save profiling data to an external file." At the bottom of the dialog box are four buttons: "< Back", "Next >", "Finish", and "Cancel".

Figure 4-116 Profiling: Selecting the destination

3. In the Profiling Filters panel (Figure 4-117), specify the filters to be used during profiling. Filters can help to limit the data collected. The filters are applied top-down, so you can have a filter that *includes* itso.ad.business before a filter that *excludes* itso.ad.*. The result of this is that no information is collected for classes in itso.ad and subpackages, but information is collected for classes in itso.ad.business and subpackages.

There are three predefined filter sets:

- **Default:** This excludes standard Java interface and implementation packages.
- **WebSphere J2EE:** This excludes some WebSphere packages in addition to the Java interface and implementation packages.
- **WebSphere Studio:** This excludes the packages defined for the WebSphere Studio Workbench.

Click **Next**.

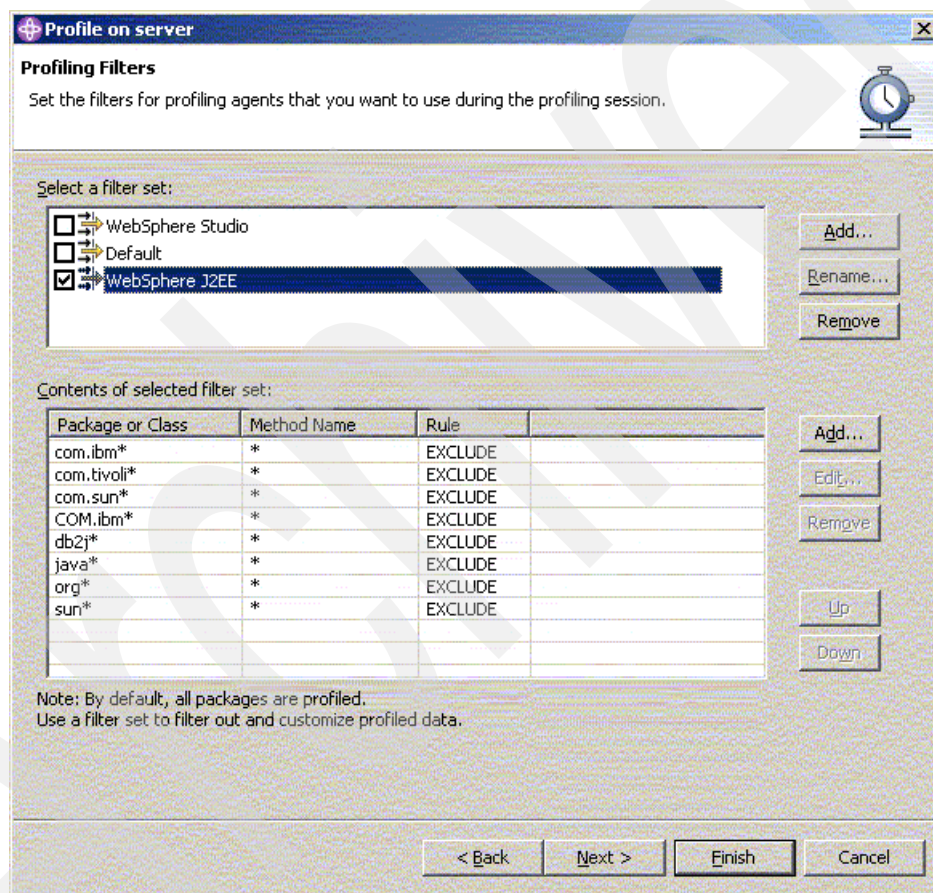


Figure 4-117 Profiling: Selecting filters

4. In the Profiling options panel (Figure 4-118), specify the profiling options. The options that you select influence the amount of information collected from profiling:
 - a. Select both **My application uses too much memory** and **My application is too slow**.
 - b. Click **Show details** for both options.
 - c. Select **Show instance level information**.
 - d. Select **Show execution flow graphical details**.
 - e. Click **Next** to continue the wizard.

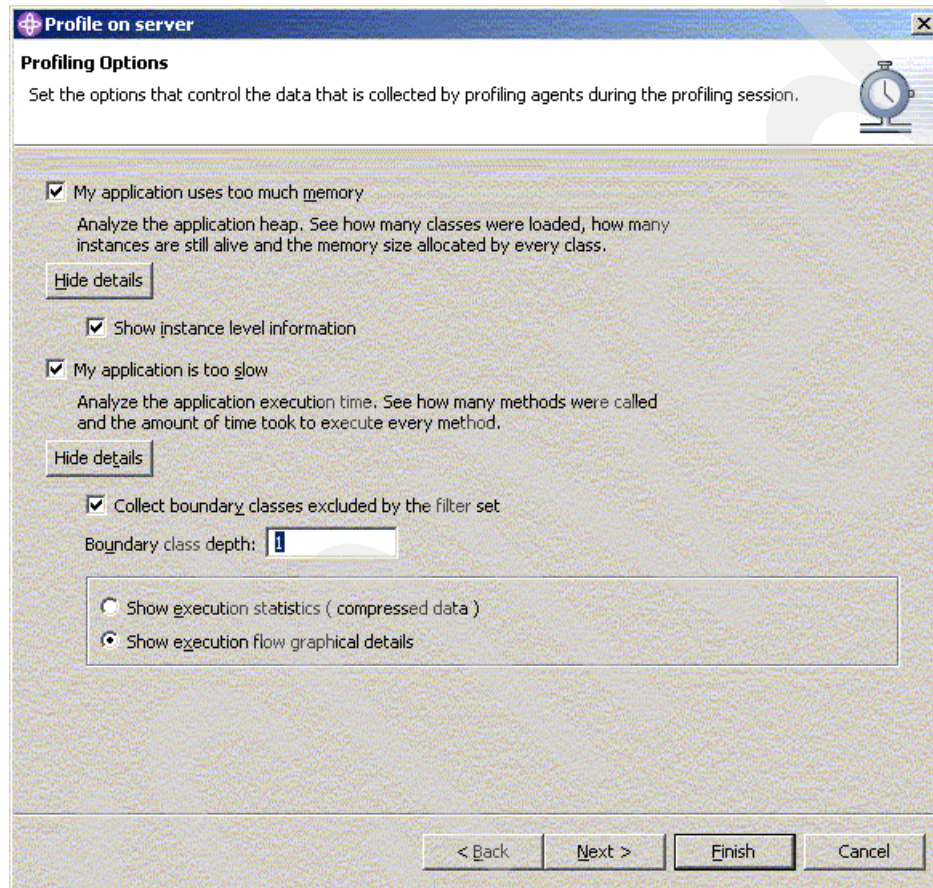


Figure 4-118 Profiling: Selecting profiling options

5. In the Profiling Limits panel (Figure 4-119), set the profiling limits. Using these limits, you can have profiling automatically stop after a number of invocations or a certain amount of time has passed. Leave both options deselected and click **Finish**.

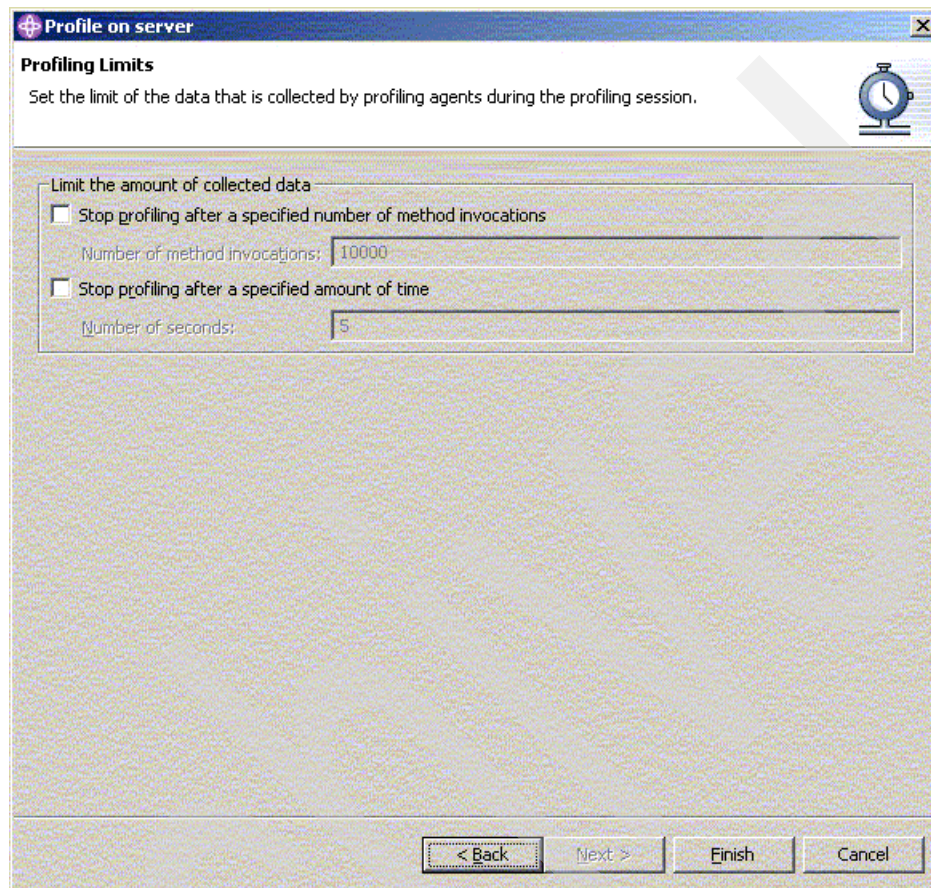


Figure 4-119 Profiling: Specifying limits

6. A window opens that shows the profiling tips. Click **OK** to close this window.

Profiling perspective

Now that the server is ready for profiling, follow these steps:

1. Access the development environment on your workstation.
2. Open the Profiling and Logging perspective within WebSphere Development Studio Client. In the Profiling and Logging perspective's Navigator, right-click your application's project. Select the **Profile on Server** option.
3. In the Profile Monitor view (Figure 4-120), right-click and select **Start Monitoring**.

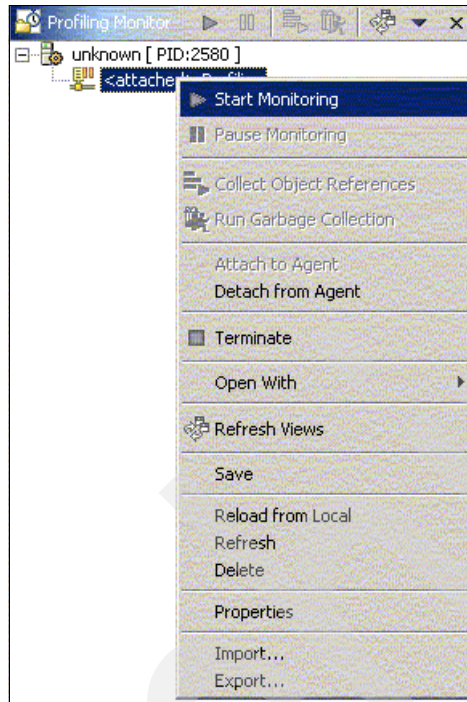


Figure 4-120 Profiling: Start monitoring

4. Begin exercising your application.
5. After you run some parts of your application, examine the collected profiling information. In the Profiling Monitor, right-click **<monitoring...>** and select **Refresh Views**.
6. You can then begin to analyze the collected data.

Analyzing timing information

After you collect the data, you can look at the runtime performance of the application.

Method statistics

Select **<monitoring...>**, right-click, and select **Open With → Method Statistics**. The Method Statistics view (Figure 4-121) opens. The timing information in this view is relevant when determining which methods are taking the most execution time. The unit of time in this view is a second.

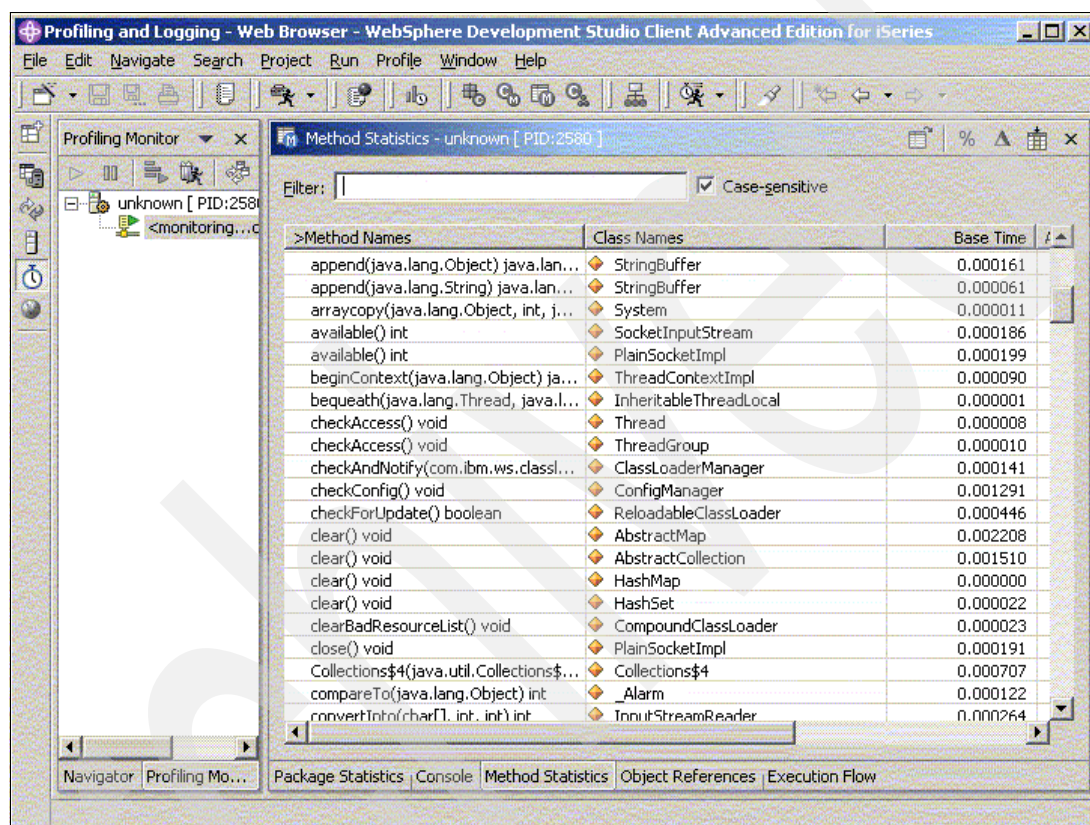


Figure 4-121 Profiling: Viewing Method Statistics

Method invocation

When you determine which methods may have a performance problem, you may want to discover the reason:

- ▶ Right-click the method and select **Show Method Invocation** (see Figure 4-122). This opens the Method Invocation view, which contains a graphical presentation of the method invocation.
- ▶ Right-click in the view and select **Show Invocation Table**. This opens the Method Invocation Table view.

The Method Invocation Table view displays each method invocation. The cumulative time presents the time spent executing the method and any methods called. You can expand the method invocation by clicking the small + in front of the method invocation. This way you can examine the timing information for methods called from the selected method.

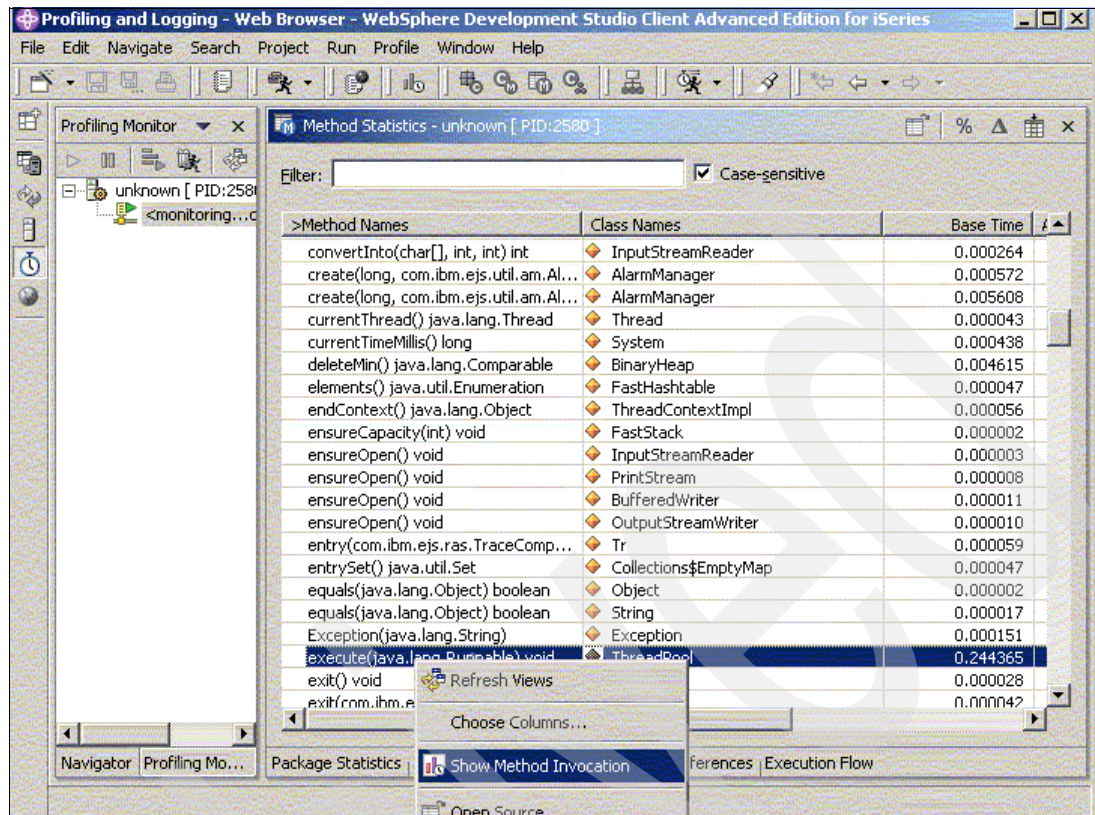


Figure 4-122 Profiling: Select Method Invocation

Examining memory utilization

Excessive memory utilization can quickly turn into a performance problem, especially if it results in memory being paged out or swapped to disk.

Instance statistics

In the Profiling Monitor, select **<monitoring...>**, right-click, and select **Open With → Instance Statistics**. This action opens the Instance Statistics view (Figure 4-123).

| >Class Names | Package | Total Instances | Live Instances | Total Size | Active Size |
|--------------------------------|--------------------|-----------------|----------------|------------|-------------|
| + PiggyBankEJBAccountTO | itso.ad.busines... | 18 | 18 | 216 | 216 |
| + PiggyBankEJBCustomerTO | itso.ad.busines... | 20 | 20 | 240 | 240 |
| + PiggyBankEJBDelegateFactory | itso.ad.busines... | 1 | 1 | 4 | 4 |
| + PiggyBankEJBDelegateImpl | itso.ad.busines... | 0 | 0 | 0 | 0 |
| + PiggyBankEJBDelegateImplTest | itso.ad.busines... | 1 | 1 | 4 | 4 |
| + SessionFacadeAdapter | itso.ad.busines... | 0 | 0 | 0 | 0 |
| + ThreadPool\$Worker | com.ibm.ws.util | 1 | 1 | 84 | 84 |

Figure 4-123 Profiling: Open the Instance Statistics view

The Instance Statistics view shows statistics that are related to instances of classes during runtime. The Total Instances column shows the number of instances that has been created during the time the profiler has been running. The Live Instances column shows the number of instances currently allocated in memory. The Total Size and Active Size columns shows the amount of memory used in bytes.

Garbage collection

To access garbage collection, follow these steps:

1. Select **<monitoring...>**, right-click, and select **Run Garbage Collection**. If the state is **<attached> Profiling**, then select **Start Monitoring** instead, because the agent must be running to perform this action.
2. Select **Profile** → **Refresh Views** from the main menu.

The result is most likely that the number of live instances has decreased for at least a couple of classes.

Object references

You can examine the object references in the running application.

1. Select **<monitoring> profiling**, right-click, and select **Collect Object References**.
2. Select **Profile** → **Refresh Views**.
3. In the Instance Statistics view, right-click and select **Show Object References**.

The Object References view opens, where you can examine the references to and from objects.

Execution flow

Select **<monitoring...> profiling**, right-click, and select **Open With** → **Execution Flow** to open the Execution Flow view. The Execution Flow view shows the interactions between classes in a graphical format. The graphical view is practical when you are trying to determine which method is taking the most processing time.

You can use the zoom icon to enlarge an interesting area. The larger the graphical view is, the more details are displayed. Each vertical bar represents a class. When you point the cursor to a vertical bar, the class name and the method that is invoked are displayed at the bottom. You can see the method names between the vertical bars. On the right side, the time line is displayed.

Right-click and select **Show Execution Table**. The Execution Flow Table view shows the detailed flow of execution from one class to another.

Other available views

Several other views are available to help you with profiling:

- ▶ **Package statistics:** Shows statistics for packages. It is possible to expand a package to see the statistics for the contained classes. This view shows object instances and can be used for memory profiling.
- ▶ **Class statistics:** Shows statistics for classes. The classes can be expanded to see the called methods. This view shows object instances.
- ▶ **Class interactions:** Shows a sequence diagram of the class interactions during execution. This can help you to understand better the execution of the application.
- ▶ **Object interactions:** Shows a sequence diagram of the object interactions during execution. This is much like the Class Interaction view except that individual objects are shown.
- ▶ **Thread interactions:** Shows the interaction between threads in the application. This view can help you profile multi-threaded applications.

- **Process interactions:** Shows the interaction between processes (JVMs). This view is available only from the Host object in the Profiling Monitor view. If you cannot see any host objects in the Profiling Monitor, click the down arrow on the title bar and select **Hosts**.

Note: You must use the J2EE Request Profiler to collect this information.

- **Host interactions:** Shows the interaction between hosts (computers). This view is only available from the Monitor object in the Profiling Monitor view. If you cannot see any Monitor objects in the Profiling Monitor, click the down arrow on the title bar and select **Monitors**.

Note: You must use the J2EE Request Profiler to collect this information.

If you want to use the Class Interactions or Object Interactions view while profiling applications running on application server, consider using the J2EE Request Profiler. This profiler hides a large part of the classes or objects that are not relevant to your application code.

Stopping the profiling session

When you finish profiling your application, you can end the session by simply stopping the server. If you want to keep the server running, perhaps to continue profiling later, you can also select **<monitoring...> Profiling**, right-click, and select **Pause Monitoring** or **Detach from Agent**.

To save the profiling data in the Profile Project, select **Save**. You can re-open the statistics views later again for further analysis.

J2EE Request Profiler Agent

J2EE Request Profiler is a special profiling agent that runs in the application server. It is useful when profiling J2EE applications. It excludes many infrastructure classes, making it easier to focus on your application classes. It also supports profiling across JVMs. This is convenient when your application is running on more than one application server. A typical scenario for using this agent is to deploy EJBs and Web applications on separate application servers.

4.5.2 Performance Monitoring Infrastructure

The Performance Monitoring Infrastructure (PMI) leverages the client-server architecture. The server collects performance data from the various WebSphere Application Server components. A client retrieves performance data from one or more servers and processes the data.

The application server collects the PMI data in memory. This data consists of counters such as servlet response time and data connection pool usage. The data points are then retrieved using a Web client, Java client, or Java Management Extensions (JMX) client. WebSphere Application Server contains the Tivoli Performance Viewer, which is a Java client that displays and monitors performance data.

The right side of Figure 4-124 shows how the server updates and keeps PMI data in memory. In the left side of the same figure, you see how a Web client, Java client and JMX client retrieve the performance data.

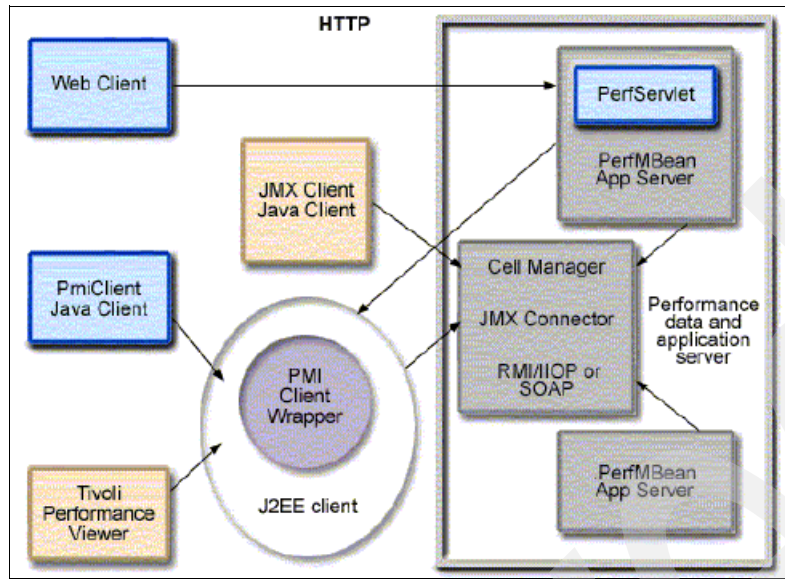


Figure 4-124 PMI: Architecture

Enabling Performance Monitor Services in WebSphere Application Server

You must enable Performance Monitor Services (PMS) in your application server instance to capture WebSphere runtime data. Use the following steps to enable PMS:

1. Access the WebSphere Application Server administrative console for the application server.
2. In the navigation tree, expand **Servers** and click **Application Servers**.
3. In the right pane, click the server for which you want to enable performance monitoring. Then click **Performance Monitoring Service**.
4. In the Performance Monitoring Service pane (Figure 4-125), complete these steps:
 - a. Click the **Configuration** tab.
 - b. Select **Startup**.
 - c. Select the desired initial specification level. You can choose the specification level for each server component. You can select None, Standard or Custom if you want a different trace level for some specific components. For Custom, you must specify any of the following levels:
 - Null=N
 - Low=L
 - Medium=M
 - High=H

The server components that are available are:

- beanModule
- cacheModule
- connectionPoolModule
- j2cModule
- jvmRuntimeModule
- orbPerfModule
- servletSessionsModule
- systemModule

- threadPoolModule
- transactionModule
- webAppModule
- webServicesModule
- wlmModule
- wsgwModule

- Click **Apply**.
- Click **Save** to save the configuration.

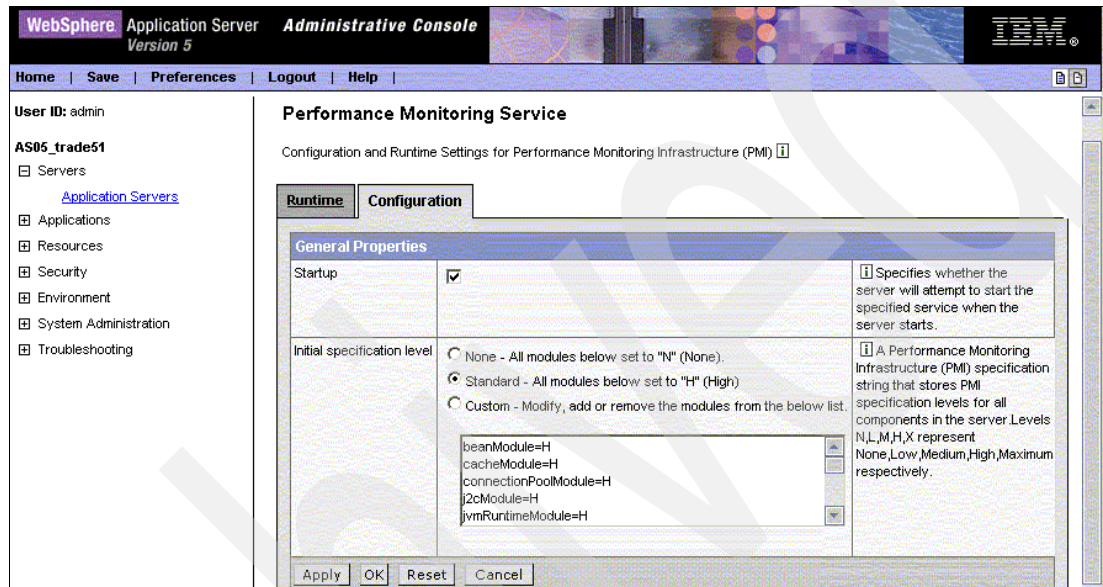


Figure 4-125 PMI: Enabling PMI data gathering

- Restart the application server. The changes made do not take effect until you restart the application server.

4.5.3 Tivoli Performance Viewer

The Tivoli Performance Viewer, known as Resource Analyzer in previous versions of the WebSphere Application Server, is a GUI-based performance monitor for WebSphere Application Server. You run Tivoli Performance Viewer on a GUI-capable workstation and connect it remotely to the WebSphere Application Server running on the iSeries. The Tivoli Performance Viewer can also monitor a locally running copy of the WebSphere Application Server on your workstation during development.

There are two ways to start the Tivoli Performance Viewer. The syntax of the command line access method is:

```
tpervviewer server port connector
```

Therefore, you may start the Tivoli Performance Viewer from the command line, as follows:

```
tpervviewer my_iseries 9105 soap
```

Alternatively, you can launch the Tivoli Performance Viewer using the icon on the desktop. Then you see a window such as the example in Figure 4-126. Enter valid values and click **OK**.



Figure 4-126 Tivoli Performance Viewer: Connecting to the application server

The Tivoli Performance Viewer is a Java client that retrieves the PMI data from an application server and displays it in a variety of formats. A wide range of performance data is available. It falls into two kinds of informational categories:

- ▶ Application resources: For example, enterprise beans and servlets
- ▶ WebSphere run-time resources: For example, JVM memory, thread pools, and database connection pools

You use the navigational pane, shown in the left side of the Tivoli Performance Viewer interface, to access available performance data. When you click the application server in the navigational pane, as shown in Figure 4-127, you can access summary performance data about the running application.

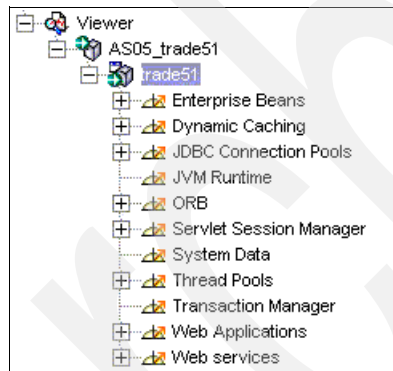


Figure 4-127 Tivoli Performance Viewer: Using the navigational pane

You can drill down into the performance data by expanding and clicking one or more areas below the application server in the navigational pane. Depending on which aspects of performance you are measuring, you can use the Tivoli Performance Viewer to accomplish the following tasks:

- ▶ View data in real time or view historical data from the log files
- ▶ View data in chart form, allowing comparisons of one or more statistical values for a given resource on the same chart

In addition, different units of measurement can be scaled to enable meaningful graphical displays.

- ▶ Record current performance data in a log and replay performance data from previous sessions
- ▶ Compare data for a single resource to a group of resources on a single node

For example, you can use the Tivoli Performance Viewer to examine the runtime of your JVM, as shown in Figure 4-128.

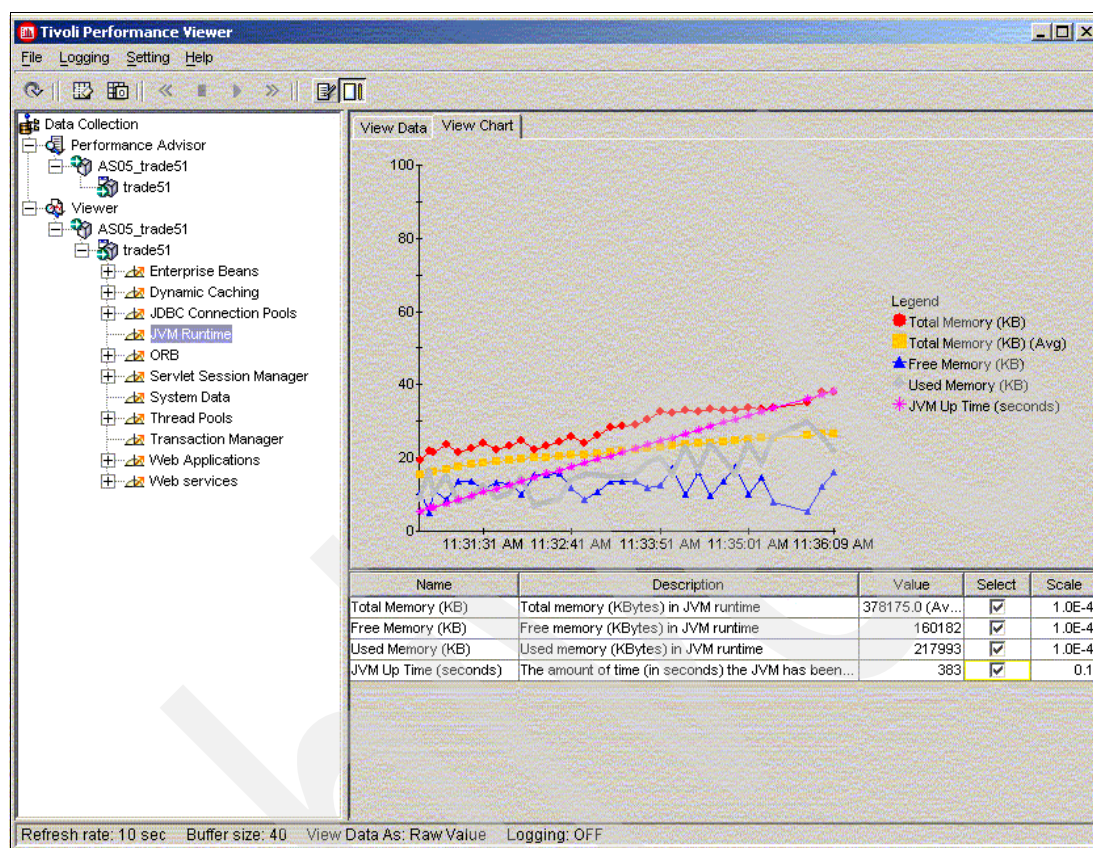


Figure 4-128 Tivoli Performance Viewer: Monitoring the JVM runtime

Using the Tivoli Performance Viewer, you can examine the available performance information, including simple counters, statistical data (such as the response time for each method invocation of an enterprise bean), and load data (such as the average size of a database connection pool during a specified time interval). This data is reported for individual resources and for multiple resources. The Tivoli Performance Viewer provides in-depth data that can help you to identify and resolve performance problems.

4.5.4 Performance Advisors

The WebSphere Application Server includes two Performance Advisors to help you tune applications and systems for optimal performance. These Performance Advisors use the PMI data to suggest configuration changes based on which settings may need to be adjusted for your environment. For more information about PMI, see 4.5.2, “Performance Monitoring Infrastructure” on page 128.

The Runtime Performance Advisor runs in the application server process. The other advisor runs in the Tivoli Performance Viewer. Table 4-4 outlines the difference between the two advisors.

Table 4-4 Performance Advisor attributes

| | Runtime Performance Advisor | Performance Advisor in Tivoli Performance Viewer |
|------------------------|--|---|
| Location of execution | Application server | Tivoli Performance Viewer client |
| Location of tool | Administrative console | Tivoli Performance Viewer |
| Output | SystemOut.log file and WebSphere run-time messages in WebSphere status panel in the administrative console | Tivoli Performance Viewer graphical user interface |
| Frequency of operation | Every 10 seconds in the background | When you select refresh in Tivoli Performance Viewer |
| Advice provided | Object Request Broker (ORB) service thread pools Web container thread pools Connection pool size Persisted session size and time Prepared statement cache size Session cache size | ORB service thread pools Web container thread pools Connection pool size Persisted session size and time Prepared statement cache size Session cache size Dynamic cache size JVM heap size DB2 Performance Configuration Wizard |

Using the Runtime Performance Advisor

The Runtime Performance Advisor runs in the JVM of the application server and periodically checks for inefficient settings. Recommendations are provided in the form of standard product warning messages. These recommendations are displayed both as warnings in the administrative console, under WebSphere Runtime Messages, and as text in the application server SystemOut.log file. The Runtime Performance Advisor has minimal system performance impact.

The Runtime Performance Advisor enables the appropriate monitoring counter levels for all enabled areas. If there are specific counters that you do not want to monitor, disable the corresponding advice in the Runtime Performance Advisor and disable unwanted counters.

Note: Before you enable the Runtime Performance Advisor, you must enable performance monitoring services for an application server and restart the application server. For WebSphere Application Server Network Deployment, you must also enable performance monitoring services for a node agent and restart the node agent.

To use the Runtime Performance Advisor, follow these steps:

1. Start the administrative console.
2. In the administrative console navigation tree, expand **Servers** and click **Application Servers**.
3. Click the name of the server for which you want to configure the Runtime Performance Advisor.
4. Click **Runtime Performance Advisor Configuration**.
5. Click the **Configuration** tab and configure the following settings (see Figure 4-129):
 - a. Select **Calculation Interval**. PMI data is taken over an interval of time and averaged to provide advice. The interval specifies the length of the time over which data is taken for this advice. Therefore, details within the advice messages appear as averages over this interval.

- b. Select the **Maximum Warning Sequence**. The maximum warning sequence refers to the number of consecutive warnings issued before the threshold is updated. For example, if the maximum warning sequence is set to 3, then the advisor only sends three warnings to indicate that a parameter should be adjusted. After that, a new alert is sent only if WebSphere Application Server reaches another threshold.
- c. Select **Number of Processors**. Select the appropriate settings for your system configuration to ensure accurate advice.

Note: A limited number of values is available. If the number of processors in your WebSphere Application Server partition is not listed, select the next highest number.

- d. Click **Apply**.
- e. Save the configuration.

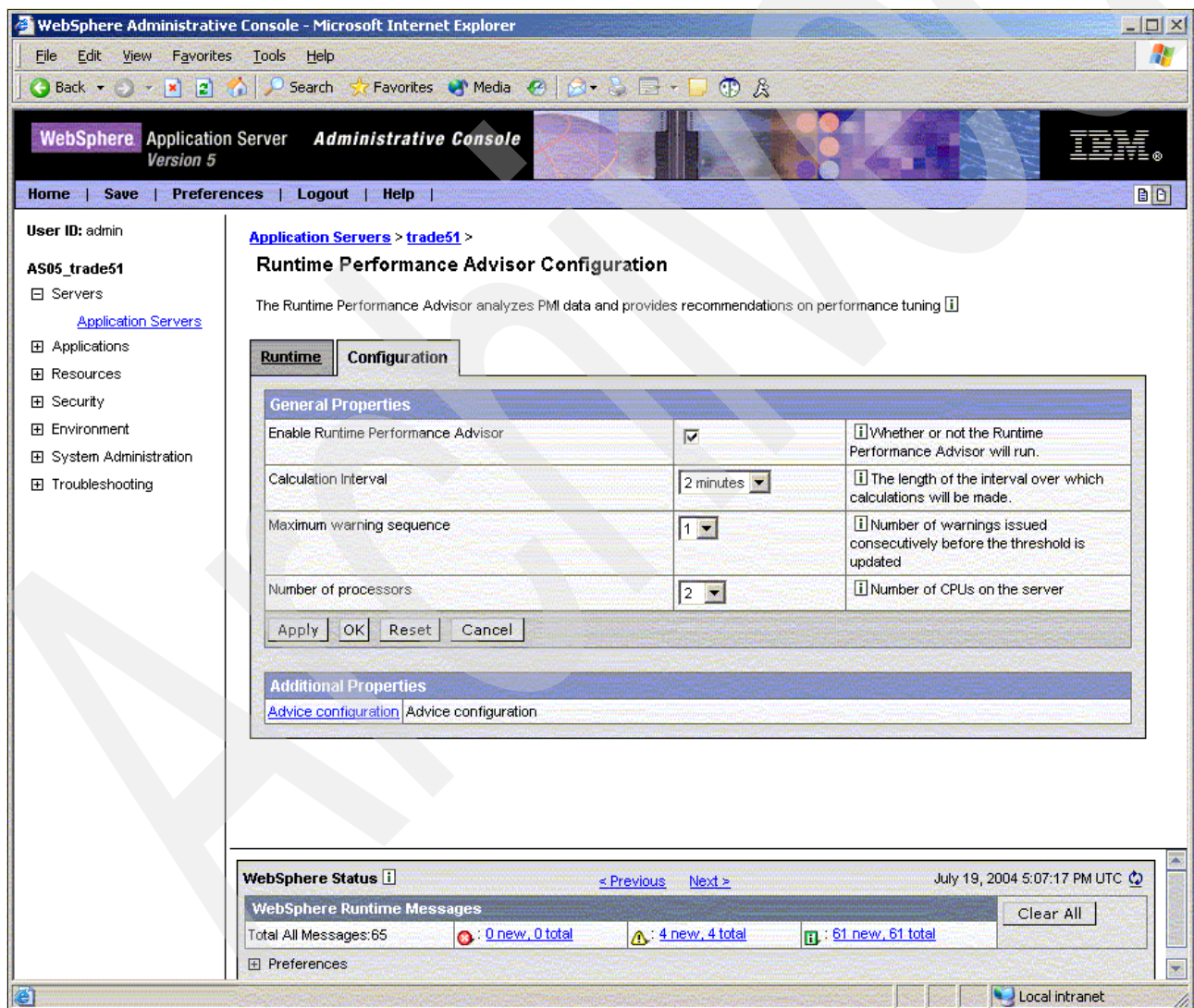


Figure 4-129 Runtime Performance Advisor: Configuration

6. Click the **Runtime** tab and click **Restart** to re-initialize the Runtime Performance Advisor based on the most recent saved configuration. This action also resets the state of the

Runtime Performance Advisor. For example, the current warning count is reset to zero for each message.

You can enable or disable the type of advice that you receive on the Advice Configuration page. To view this administrative page, complete these steps:

1. Click **Servers** → **Application Servers** → *server name* → **Runtime Performance Advisor Configuration** → **Advice Configuration**.
2. In the Advice Configuration panel, click the **Configuration** tab. Note these columns (Figure 4-130):
 - **Advice name:** Specifies the advice that you can enable or disable.
 - **Advice applied to component:** Specifies the WebSphere Application Server component to which the runtime performance advice applies.
 - **Advice status:** Specifies whether advice is stopped or started. There are only two values: Started and Stopped. *Started* means that the advice runs if the advice applies. *Stopped* means that the advice does not run.

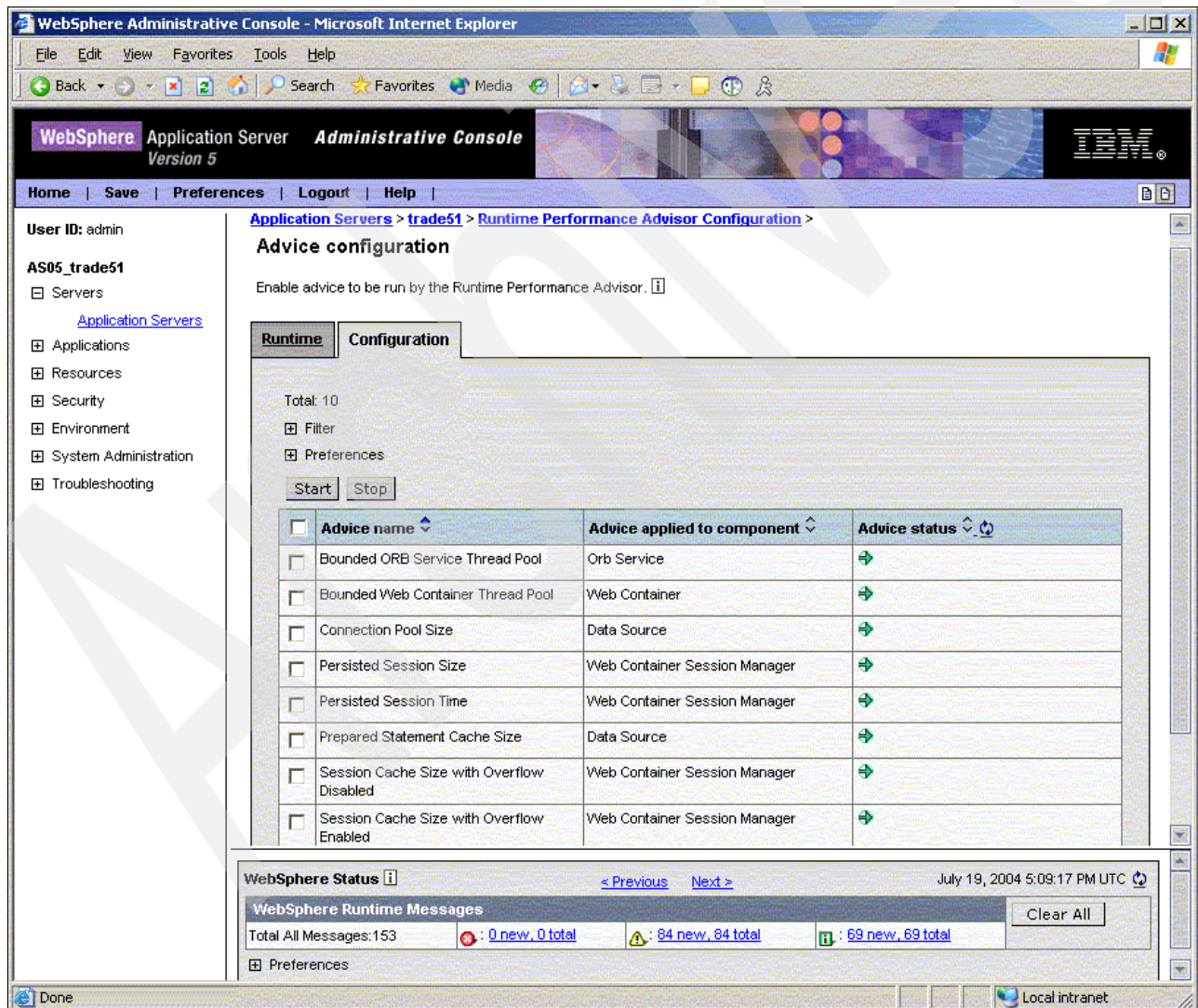


Figure 4-130 Runtime Performance Advisor: Advice Configuration

3. Click the **Runtime** tab. You see these columns:

- **Advice name:** Specifies the advice that you can enable or disable.
- **Advice applied to component:** Specifies the WebSphere Application Server component to which the runtime performance advice applies.
- **Advice status:** Specifies whether advice is stopped, started, or unavailable. *Started* means that the advice is being applied. *Stopped* means that the advice is not applied. *Unavailable* means that the advice does not apply to the current configuration, such as Persisted Session Size advice in a configuration without persistent sessions.

Simulate a production-level load test. If you are using the Runtime Performance Advisor in a test environment, or doing any other tuning for performance, simulate a realistic production load for your application. The application should run the load test without errors. This test simulation should include the number of concurrent users that match peak periods as well as stress testing system resources, such as CPU and memory to the levels expected in production. Some advice may not be accurate if sufficient load is not generated.

To view the tuning advice following the stress test, use one of these two options:

- In the administrative console, select **Warnings** under the WebSphere Runtime Messages in the WebSphere Status panel.
- View the SystemOut.log file.

Figure 4-131 shows the runtime events from the Runtime Performance Advisor by viewing the Warnings in the WebSphere status panel.

http://rchasm05:9110/admin/eventCollection.do?previousAction=<+Previous - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Home Search Favorites Media

Runtime Events

Runtime events propagating from the server ⓘ

Total: 166 Page: 1, Total Pages: 9 < Previous Next >

Filter Preferences

| Timestamp | Message Originator | Message |
|-----------------------------|---|---|
| Jul 19, 2004 5:12:38 PM UTC | com.ibm.ws.performance.tuning.serverAlert.TraceResponse | TUNE0204W: Decreasing the size of the Orb Service |
| Jul 19, 2004 5:12:38 PM UTC | com.ibm.ws.performance.tuning.serverAlert.TraceResponse | TUNE0204W: Decreasing the size of the Web Container |
| Jul 19, 2004 5:12:38 PM UTC | com.ibm.ws.performance.tuning.serverAlert.TraceResponse | TUNE0214W: The session cache for trade3#trade3Web |
| Jul 19, 2004 5:15:58 PM UTC | com.ibm.ws.performance.tuning.serverAlert.TraceResponse | TUNE0214W: The session cache for trade3#trade3Web |
| Jul 19, 2004 5:16:46 PM UTC | com.ibm.ws.performance.tuning.serverAlert.TraceResponse | TUNE0214W: The session cache for trade3#trade3Web |
| Jul 19, 2004 5:08:20 PM UTC | com.ibm.ws.naming.util.Helpers | NMSV0605W: A Reference object looked up from the c |
| Jul 19, 2004 5:08:20 PM UTC | com.ibm.ws.naming.util.Helpers | NMSV0610I: A NamingException is being thrown from |
| Jul 19, 2004 5:08:20 PM UTC | com.ibm.ws.naming.util.Helpers | NMSV0605W: A Reference object looked up from the c |
| Jul 19, 2004 5:11:27 PM UTC | com.ibm.ws.performance.tuning.serverAlert.TraceResponse | TUNE0214W: The session cache for trade3#trade3Web |
| Jul 19, 2004 5:11:49 PM UTC | com.ibm.ws.performance.tuning.serverAlert.TraceResponse | TUNE0214W: The session cache for trade3#trade3Web |
| Jul 19, 2004 5:12:00 PM UTC | com.ibm.ws.performance.tuning.serverAlert.TraceResponse | TUNE0214W: The session cache for trade3#trade3Web |
| Jul 19, 2004 5:14:50 PM UTC | com.ibm.ws.performance.tuning.serverAlert.TraceResponse | TUNE0214W: The session cache for trade3#trade3Web |
| Jul 19, 2004 5:08:21 PM UTC | com.ibm.ws.naming.util.Helpers | NMSV0610I: A NamingException is being thrown from |
| Jul 19, 2004 5:08:21 PM UTC | com.ibm.ws.naming.util.Helpers | NMSV0605W: A Reference object looked up from the |

http://rchasm05:9110/admin/eventCollection.do?EditAction=true&refId=36&contextId=na&resourceUri=warnings&perspective=tab.o Local intranet

Figure 4-131 Runtime Performance Advisor: Viewing the runtime events

After viewing the list of runtime events, you may want to examine each item in the list in more detail. Click any of the messages in the list to see the message details display for the message, as shown in Figure 4-132.

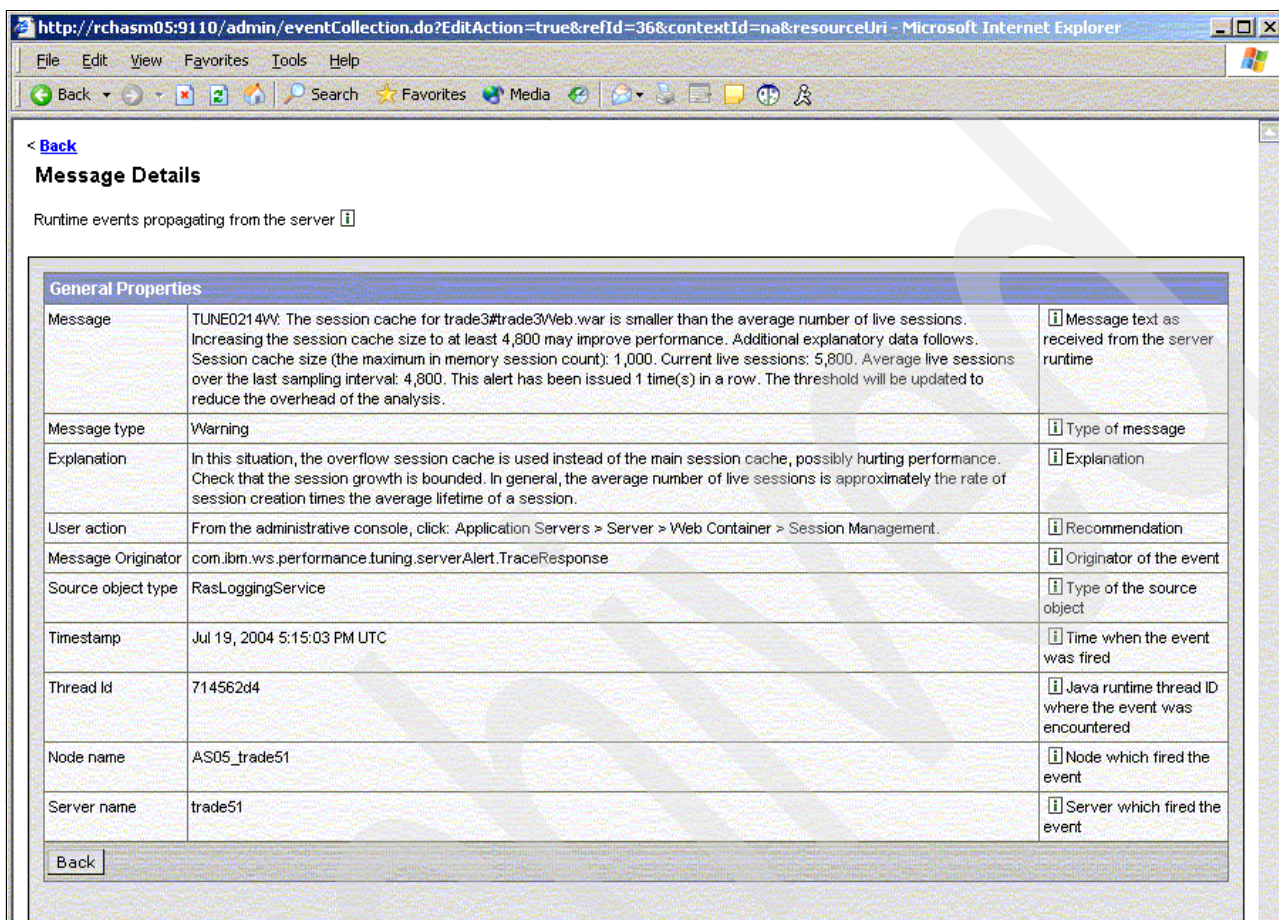


Figure 4-132 Runtime Performance Advisor: Viewing advice details

Using the second option for viewing Runtime Performance Advisor data, the SystemOut.log file, provides an easy mechanism to view the advice following the stress test. Advice details are listed together with additional explanatory data, as shown in Figure 4-133.

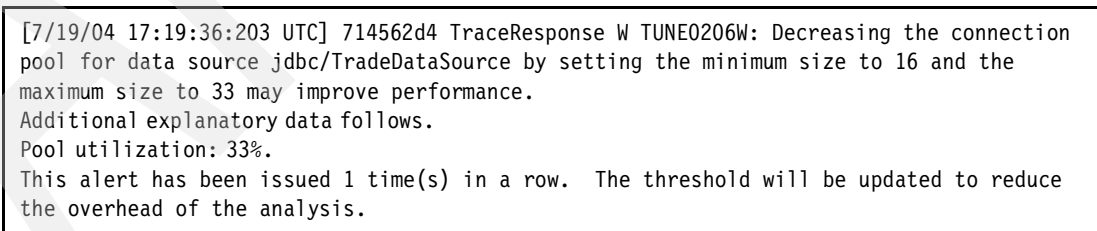


Figure 4-133 Runtime Performance Advisor: Viewing the advice in the SystemOut log file

Note: Some messages may not be issued immediately.

Analyze the advice. Make changes to your configuration if necessary. Although the Performance Advisor attempts to distinguish between loaded and idle conditions, the advisor may provide misleading advice if it is enabled when the system is not in one of those states.

This result is especially likely when you run short tests. Although the advice helps in most configurations, there may be situations where the advice hinders performance. Due to these conditions, advice is not guaranteed. Therefore, test the environment with the updated configuration to verify that it functions and performs well.

Using the Performance Advisor in the Tivoli Performance Viewer

The Performance Advisor in the Tivoli Performance Viewer analyzes collected PMI data and provides advice to help tune systems for optimal performance. It also provides more extensive advice than the Runtime Performance Advisor. For example, the Performance Advisor in Tivoli Performance Viewer provides advice on setting the dynamic cache size and setting the JVM heap size.

To enable and run the Performance Advisor, follow these steps:

1. Enable performance monitoring services for an application server and restart the application server. See 4.5.2, “Performance Monitoring Infrastructure” on page 128, for additional information.
2. (Optional) If you are running WebSphere Application Server Network Deployment, enable performance monitoring services for a node agent and restart the node agent.
3. Enable data collection with the administrative console:
 - a. Start the administrative console.
 - b. In the navigation tree, expand **Servers** and click **Application Servers**.
 - c. Click the name of the server for which you want to enable data collection.
 - d. Click the **Runtime** tab.
 - e. Click **Performance Monitoring Service**.
 - i. Select the PMI modules and levels to set the initial specification level field.
 - ii. Click **Apply** or **OK**.
 - f. Save the configuration.

These changes take effect immediately, but are not persistent. Use the **Configuration** tab to make persistent changes.

4. Start Tivoli Performance Viewer.
 - a. If you are using a Windows-based workstation, click **Start** → **Programs** → **IBM WebSphere** → **Application Server** → **Tivoli Performance Viewer**. In Windows, the Tivoli Performance Viewer uses the settings in *tperfviewer.bat* to connect to your application server. If you want to change these settings, see “Changing the default settings in *tperfviewer.bat*” on page 140.
 - b. You can also start Tivoli Performance Viewer from a command prompt:
 - i. Open a command prompt.
 - ii. Use the **cd** command to change to the `product_installation_directory\bin` directory.
 - iii. Run the *tperfviewer* batch file:

```
tperfviewer.bat [host_name] [ port_number ] [ connector_type ]
```

Here *host_name* is the iSeries host name, *port_number* is the Simple Object Access Protocol (SOAP) or Remote Method Invocation (RMI) connector port for the application server from which you are collecting data, and *connector_type* is the type of connector to use. The default value for *port_number* is 8880 for WebSphere Application Server and 8879 for WebSphere Application Server Network Deployment. Valid values for *connector_type* are SOAP or RMI. The default connector type is SOAP.

Changing the default settings in tperviewer.bat

You can change the default settings in the tperviewer.bat file so that Tivoli Performance Viewer connects to your application server. To change the settings, follow these steps:

1. On your workstation, open C:\<product_installation_directory>\bin tperviewer.bat in a text editor. The default installation directory is C:\Program Files\WebSphere\AppServer.
2. Under the :LOCAL entry, locate the following lines:

```
set DEST=localhost
set DESTPORT=8879
```

Replace the lines with these lines:

```
set DEST=host_name
set DESTPORT=port_number
```

Here *host_name* is the name of your iSeries server, and *port_number* is the SOAP port for your application server.

These steps assume that you are using the SOAP connector. If you use the RMI connector, specify the application server's RMI port number. Locate the following line:

```
set CONNECTOR=SOAP
```

Replace the previous line with this one:

```
set CONNECTOR=RMI
```

3. After you start Tivoli Performance Viewer, you can perform any of these tasks:
 - Set performance monitoring levels for resources
 - View performance data based on the current settings
 - Save performance data to a log file
4. Simulate a production level load by running a stress test of the application. If you are using the Performance Advisor in a test environment, or doing any other performance tuning, simulate a realistic production load for your application. The application should run this load without errors. This simulation should include the number of concurrent users that is typical of peak periods, and drive system resources such as CPU and memory to the levels expected in production. Some advice may not be accurate if sufficient load is not generated.
5. Select the Advisor icon to display tuning advice. Double-click a message for details. Because PMI data is taken over an interval of time and averaged to provide advice, details within the advice message appear as averages.
6. (Optional) Refresh the data. You can refresh data using one of two ways:
 - Select your application server in the navigation tree and click **Refresh**. Tivoli Performance Viewer queries the server for new PMI and product configuration information.
 - Select your application server under **Performance Advisor** → *node_name*, and then click **Refresh**. Tivoli Performance viewer refreshes advice that is provided at a single instant in time, but does not query the server for new PMI and product configuration information.
7. Update the product configuration based on the advice.

Because Tivoli Performance Viewer refreshes advice at a single instant in time, take the advice from the peak load time. Although the Performance Advisor attempts to distinguish between loaded and idle conditions, it may provide misleading advice if it is enabled when the system is ramping up or down. This result is especially likely during short tests. Although the advice helps in most configurations, there may be situations where the advice hinders

performance. Due to these conditions, advice is not guaranteed. Therefore, test the environment with the updated configuration to verify that it functions and performs well.

Performance Advisor Report in Tivoli Performance Viewer

You can view the recommendations and data from the Performance Advisor in Tivoli Performance Viewer. You do this by expanding the Performance Advisor icon under Data Collection in Tivoli Performance Viewer and selecting the server. You can view the detailed advice messages as well as performance data.

After you view a message in the upper right pane of the Performance Advisor, double-click the message to obtain more details, as shown in Figure 4-134.

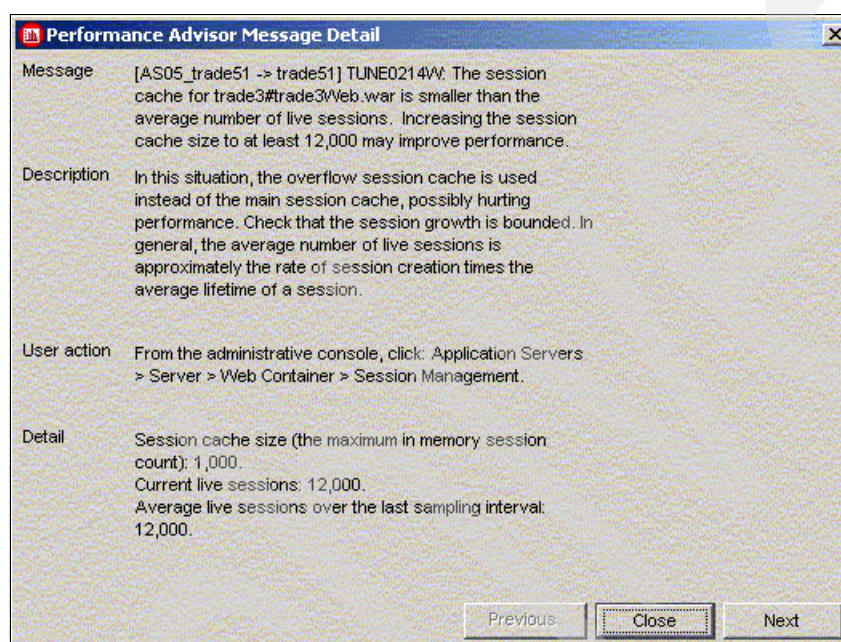


Figure 4-134 Tivoli Performance Viewer's Performance Advisor: Message details

In the lower right pane of the Performance Advisor, you can view a summary of performance data for the WebSphere Application Server. The data here corresponds to the same period that recommendations were provided for. However, recommendations may use a different set of data points during analysis than the set displayed by the summary page.

Figure 4-135 shows the data captured in the Performance Advisor. The first table represents the number of requests per second and the response time in milliseconds for both the Web and EJB containers. The pie graph displays the CPU activity as percentage busy and idle. The bar graphs display average thread activity for the Web container and ORB thread pools, and average database connection activity for connection pools. Activity is expressed as the number of threads/connections busy and idle.

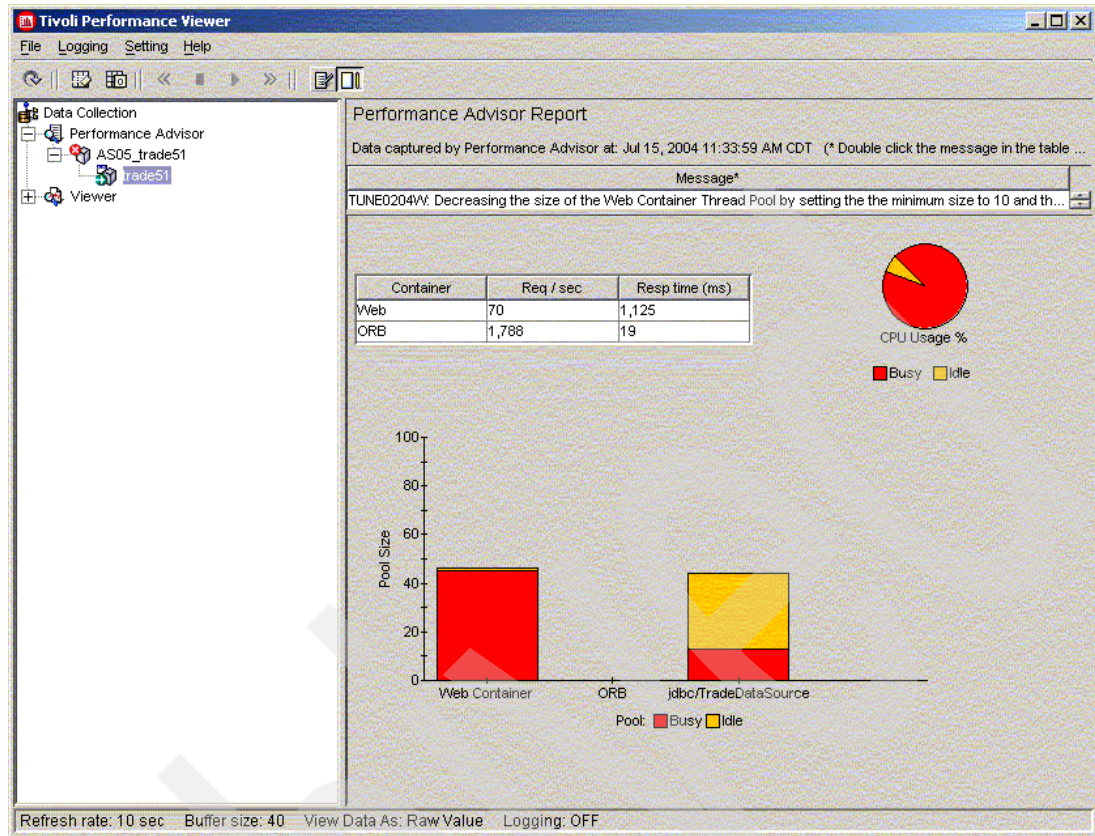


Figure 4-135 Tivoli Performance Viewer's Performance Advisor: Performance data

4.5.5 Log Analyzer

The Log Analyzer is a GUI tool that permits the user to view any logs generated with Log Analyzer TraceFormat, such as the IBM service log file and other traces. It can take one or more service logs or trace logs, merge all the data, and display the entries in sequence.

More importantly, this tool is shipped with an XML database (the symptom database), which contains recommendations for some common problems, reasons for the errors, and recovery steps (see Figure 4-136). The Log Analyzer compares every error record in the log file to the internal set of known problems in the symptom database and displays all the matches. This allows a user to receive error message explanations and information such as why the error occurred and how to recover from it.

Obtaining the Log Analyzer

The Log Analyzer is installed on your workstation as part of the WebSphere Application Server workstation components.

Using the Log Analyzer

Follow these steps to use the Log Analyzer:

1. You can start the Log Analyzer on your workstation by using either of these methods:
 - From the Windows task bar, select **Start → Programs → IBM WebSphere → Application Server Version 5.1 → Log Analyzer**.
 - From the workstation command prompt:
 - On Windows workstations, enter:
`[WAS_INSTALL_ROOT]\bin\waslogbr.bat`
 - On UNIX platforms, enter:
`[WAS_INSTALL_ROOT]/bin/waslogbr.sh`
2. In the Log Analyzer window, perform *either* of these actions to open an activity log file:
 - Use File Transfer Protocol (FTP) to send the activity.log file in binary format to your Log Analyzer workstation.
 - On Windows workstations, map a drive to the iSeries server. Alternatively, on UNIX platforms, mount a drive to the iSeries server.
3. Select **File → Open**.
4. Navigate to the directory that contains the activity.log file. Select the **activity.log** file and click **Open**.
5. Expand the tree in the left pane of the Log Analyzer to view messages.

As illustrated in Figure 4-136, records without any color are considered *normal*. Records that are highlighted in yellow indicate warnings. And records that are highlighted in pink indicate errors. If you select a record, you can view its contents, including the basic error or warning message, the date, the time, which WebSphere component logged the record, and which process (for example, application server or node agent) it came from, in the right pane.

The activity log does not analyze any other log files, such as the SystemOut.log or native_stdout.log file.

To analyze the records, right-click a record in the tree on the left (click the UnitOfWorkView at the top to select them all), and click **Analyze**. Any records with a green checkmark next to them match a record in the symptom database. When you select a record that has a checkmark, you see an explanation of the problem in the lower right pane (see Figure 4-136).

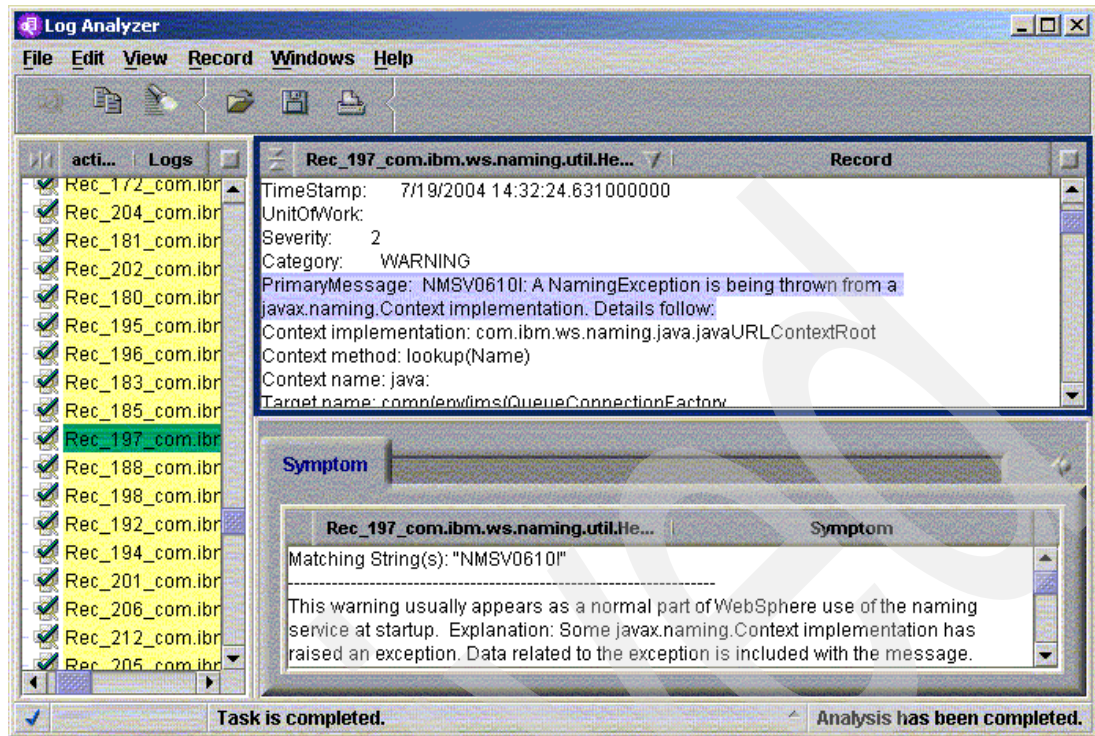


Figure 4-136 Log Analyzer: Viewing log details

Updating the symptom database

The database of known problems and resolutions is periodically enhanced as new problems arise and new versions of the product are introduced. To ensure that you have the latest version of the database, select **File** → **Update Database** → **WebSphere Application Server Symptom Database** from within the Log Analyzer tool, at least once a month. Users who have just installed the product and have never run the update should do this immediately, since updates may have occurred since the version was first released.

The knowledge base used for analysis is built gradually from problems that are reported by users. For a recently released version of the product, you may not find any analysis hits. However, the Log Analyzer tool still provides a way to see all error messages and warnings from all components in a convenient, user-friendly way.

Viewing the IBM Service log without the Log Analyzer

If using the Log Analyzer to view the IBM Service log file is impractical or inconvenient, you can use an alternative tool called **showlog**. To use the **showlog** tool, perform these steps:

1. On the iSeries command line, run the Start Qshell Interpreter (STRQSH) command.
2. Change to the bin directory for the product:

```
cd was_install_root/bin
```

Here *was_install_root* is /QIBM/ProdData/WebAS51/Base for WebSphere Application Server or /QIBM/ProdData/WebAS51/ND for WebSphere Application Server Network Deployment V5.1.

3. Run the **showlog** script. The syntax is:

```
showlog [-instance instance_name] activity_log_file [output_file]
```

Note the following explanation of the parameters:

- -instance is optional. The value *instance_name* specifies the name of the WebSphere Application Server instance. The default value for *instance_name* is default.
- - activity_log_file is required. The qualified integrated file system (IFS) path name of the activity log file that you want to view is, for example, /QIBM/UserData/WebAS51/Base/default/logs/activity.log
- - output_file is optional. The qualified IFS path name of the file to contain the formatted, readable contents of the activity log file. If this parameter is not specified, the output is written to standard out (the screen).

Consider the following examples:

```
showlog /QIBM/UserData/WebAS51/Base/default/logs/activity.log
```

```
showlog -instance myinst /QIBM/UserData/WebAS51/ND/myinst/logs/activity.log  
/home/myuserid/myinst_activity.txt
```

The output of the **showlog** script file consists of message event entries separated by horizontal lines, as shown in Figure 4-137.

```
$LANG = en_US  
$CODESET = ISO8859_1  
-----  
ComponentId: Application Server  
ProcessId: 013198/QEJBVR/TRADE51  
ThreadId: d5db3d7c  
SourceId: com.ibm.ejs.container.util.ExceptionUtil  
ClassName:  
MethodName:  
Manufacturer: IBM  
Product: WebSphere  
Version: Platform 5.1 [CLIENT 5.1.0 b0344.02] [BASE 5.1.0.4 cf40420.03] [ND 5.1.0.4  
cf40420.03]  
ServerName: AS05_trade51\AS05_trade51\trade51  
TimeStamp: 2004-07-19 19:22:49.129000000  
UnitOfWork:  
Severity: 1  
Category: ERROR  
PrimaryMessage: CNTR0020E: Non-application exception occurred while processing method  
"login" on bean "BeanId(trade3#trade3EJB.jar#TradeEJB, null)". Exception data:  
javax.ejb.TransactionRolledbackLocalException: ; nested exception is:  
javax.transaction.TransactionRolledbackException: Transaction is ended due to timeout
```

Figure 4-137 Log Analyzer: Viewing the output of the showlog script

4.6 HTTP performance tools

Performance concerns are not limited to the Java and WebSphere layers. Quite often you may experience an issue in the database layer. See 4.2.8, "Database Monitor for iSeries" on page 85, and 4.2.10, "SQL Visual Explain" on page 91, for help with identifying and resolving database-related concerns. Or you may have an issue with HTTP and the communications layer. This section examines some of the tools that you may use to diagnose and resolve HTTP-related issues.

4.6.1 Server logging

Server logs are most useful for monitoring server activity and keeping track of user access as well as being a valid aid in debugging performance issues. By carefully examining these logs, you can discover the root cause when you experience an HTTP-related problem.

Figure 4-138 shows the options that you can set for HTTP logging.

Logging ?

General Settings Custom Formats Custom Logs

User Tracking (Cookies) Custom Environment Variables

Error Logs Script Logs FRCA Logs

Enable error logging: Enabled ?

Error log: logs/error_log ?

Expiration: 1 Weeks or... ?

Maximum cumulative size: 0 Bytes or... ?

Error log format: Standard ?

Error log entries: ?

Logging level: Warning

- ✓ Emergency
- ✓ Alert
- ✓ Critical
- ✓ Error
- ✓ Warning

Figure 4-138 HTTP performance: Server logging configuration

Error logs

HTTP servers created using browser-based wizard on the HTTP Administration page always produce an error log by default. Look for error log files in the /logs subdirectory of your server root. Basic error logs are most useful for debugging configuration problems, such as when a document is not accessible or the Uniform Resource Identifier (URI) (the path you add after the server address) you're pointing to is not working as expected.

Error logs also keep track of configuration changes and server end and restart. Plus they record some system errors. Be aware that any problem detected after server initialization is most likely not recorded in the server's QTMHHTTP job logs (unless a critical condition occurs), but in the error log.

Figure 4-139 shows an example of what you may see in the error log of your HTTP server.

```
[Mon Jul 19 16:24:47 2004] [notice] Initializing the WebSphere Plugin
[Mon Jul 19 16:24:52 2004] [notice] Initializing the WebSphere Plugin
[Mon Jul 19 16:25:37 2004] [notice] Initializing the WebSphere Plugin
[Mon Jul 19 16:26:20 2004] [notice] ZSRV_MSG0385: Apache configured -- resuming normal
operations.
```

Figure 4-139 HTTP performance: Error logging

Access logs

If you choose to configure an access log for your HTTP server, information is recorded about every access attempted. By default, the access log is stored in the /logs subdirectory of your server root. Figure 4-140 shows an example of an entry from the access log.

```
192.168.1.25 - - [19/Jul/2004:16:38:00 +0000] "GET /snoop HTTP/1.1" 200 15672 "-"
"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322)"
```

Figure 4-140 HTTP performance: Access logging

Let's dissect the access log entry in Figure 4-140. In Table 4-5, you can see that the first column contains each of the elements that make up the access log entry in Figure 4-140. The tokens in the second column are used to configure the logging format in the HTTP configuration. The last column provides some detail about what each entry means.

Table 4-5 Understanding the HTTP access log

| Log entry | Token | Meaning |
|---|---------------|---|
| 192.168.1.25 | %h | The remote host IP address. |
| - | %l | The user logged on to the remote system In this case, the browser did not provide the users name for security reasons. |
| - | %u | Name of the authenticated user No user authentication is required yet. |
| 19/Jul/2004:16:38:00 +0000 | %t | Date and time the request was served. |
| "GET /snoop HTTP/1.1" | %r | The request received We can identify the method (GET), the URI and the protocol used for this transaction (HTTP version 1.1). |
| 200 | %>s | The HTTP status code 200 means that this transaction was successful. See Table 4-6 for more information about HTTP status codes. |
| 15672 | %b | The amount of data transmitted, in bytes |
| "-" | %{Referer} | The page we came from There is none in this case, since we manually typed our address. |
| "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322)" | %{User-Agent} | Information about the Web browser and operating system in use by the client |

The error and access logs are but two of the available types of logging. You may also want to track other activities, such as script interaction, to isolate and resolve performance concerns.

For detailed information about the available logging options and configuration steps, refer to:

- ▶ *HTTP Server (powered by Apache): An Integrated Solution for IBM iSeries Servers*, SG24-6716
- ▶ IBM HTTP Server for iSeries Information Center
http://publib.boulder.ibm.com/pubs/html/iseriess_http/v5r1/info/rzaie/rzaie1logformat.htm

You may also want to review the Worldwide Web Consortium (W3C) log file standards at:

<http://www.w3.org>

HTTP status codes

As discussed in 4.6.1, “Server logging” on page 146, the HTTP server logs contain a wealth of useful problem diagnosis information. In particular, the access log records the HTTP status of each request that is made. Table 4-6 shows the highest level status code groups.

Table 4-6 HTTP status codes

| Status code group | Meaning | Example |
|-------------------|---|-------------------------|
| 1xx | Informational: This contains a provisional response that influences the client's behavior. | 101 Switching protocols |
| 2xx | Successful: This also returns information about the status of negotiations and transactions. | 200 OK |
| 3xx | Redirection: This requests an action from the user agent. | 304 Not Modified |
| 4xx | Client error: These errors are sent to the user's browser when a request cannot be served. | 404 Not Found |
| 5xx | Server error: These errors are sent to the user's browser when the server cannot fulfill the request due to an internal problem. | 503 Service Unavailable |

For more detailed information about HTTP status codes and their meaning, refer to:

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10>

Aside from viewing available logging to identify and resolve performance concerns, these same logs can be useful for examining trends, such as growth in application access over time.

There are several tools, both open source and commercial, that can be used to analyze HTTP logs. One such commercial tool is the IBM Tivoli Web Site Analyzer. You can learn more about the IBM Tivoli Web Site Analyzer at:

<http://www.ibm.com/software/tivoli/products/web-site-analyzer/>

Another open source tool is *AWStats*. This free Log Analyzer generates graphical representations of Web, FTP, or mail server statistics, based upon log data. AWStats can work as a Common Gateway Interface (CGI) or from command line.

It shows you all the possible information that your log contains, in a few graphical Web pages. It uses a partial information file to process large log files quickly. It can analyze HTTP log files (NCSA combined/XLF/ELF log format or common/CLF log format) as well as FTP and mail server log files. In Figure 4-141, you see that AWStats can provide an at-a-glance picture of the types of operating systems and browsers that are accessing your application.

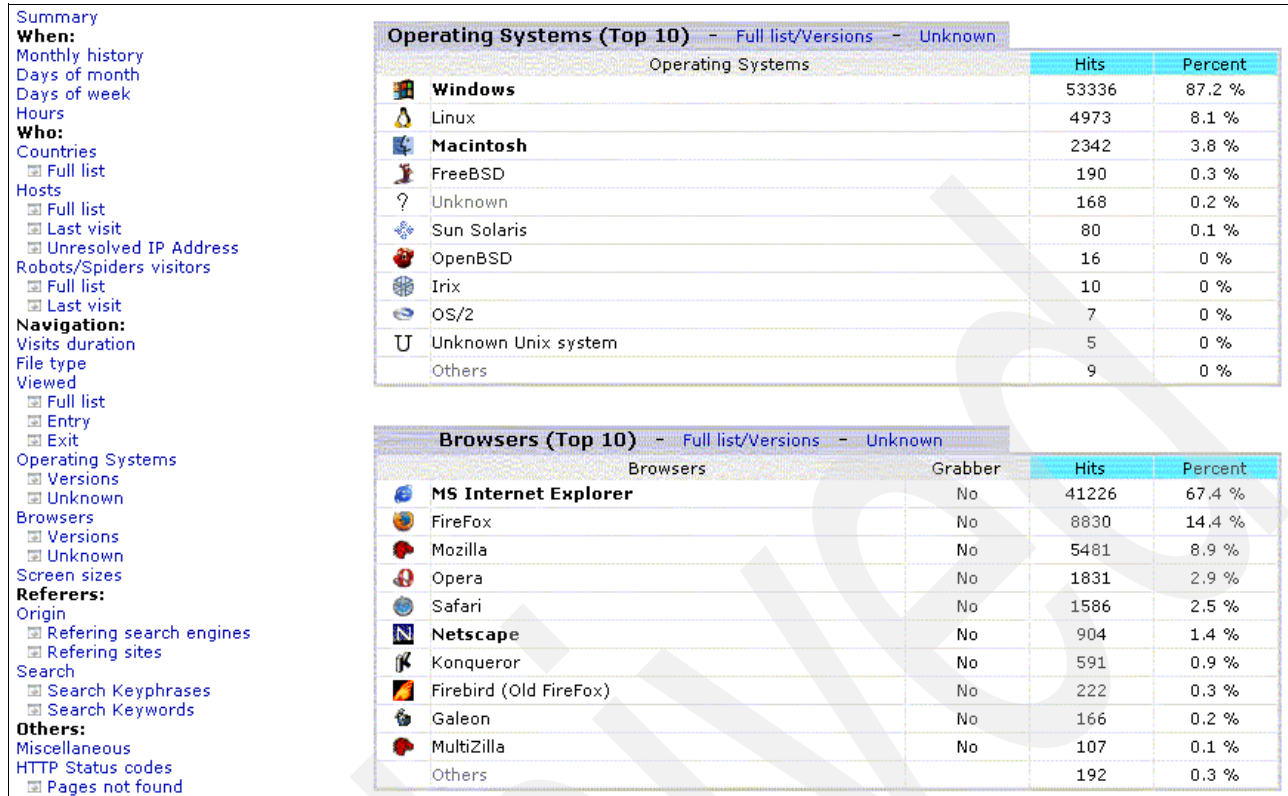


Figure 4-141 HTTP Performance: AWStats analyzes HTTP log data

For more information about the AWStats Log Analyzer, see:

<http://awstats.sourceforge.net>

4.6.2 HTTP server tracing

HTTP server tracing provides additional information about server operations, including process management and URI interpretation. If you believe your performance issue exists in the HTTP layer, tracing can provide the level of detail you need to isolate and resolve the issue. Generally speaking, you want to turn on tracing, recreate the performance problem, turn off tracing, and then review the trace output for more information.

If your HTTP server is not started, a server trace can be activated from the browser-based HTTP administration interface (<http://myserver:2001>) using the Manage all HTTP Servers display. Before you click the start icon, enter either the -ve, -vi, or -vv switch in the startup parameters field. Table 4-7 on page 152 discusses the available startup switches.

The Start TCP/IP Server (STRTCPSVR) command also supports the same startup options. If your HTTP server is down, you can start it using STRTCPSVR and specify the startup switch, as shown in Figure 4-142.

```

                                Start TCP/IP Server (STRTCPSVR)

Type choices, press Enter.

Server application . . . . . > *HTTP          *ALL, *AUTOSTART...
      + for more values
Restart server . . . . . *NONE          *NONE, *HTTP, *DNS, *DHCP...
HTTP server:
  Server instance . . . . . MYSERVER      Name, *ALL, *ADMIN
  Instance startup values . . . -vv

```

Figure 4-142 HTTP tracing: Starting a verbose trace using the STRTCPSVR command

When your HTTP server completes the startup process, check the job log for the HTTP server jobs (QTMHHTTP) to verify that the trace has been successfully activated, as shown in Figure 4-143.

```

                                Additional Message Information

Message ID . . . . . : CPCA984      Severity . . . . . : 00
Message type . . . . . : Completion
Date sent . . . . . : 07/20/04      Time sent . . . . . : 11:36:23

Message . . . . . : User Trace option changed for job
                   011472/QTMHHTTP/MYSERVER.

```

Figure 4-143 HTTP tracing: Verifying that tracing is activated

With tracing activated, you can recreate the performance problem. For example, if you are noticing poor performance in a test environment, you can recreate the issue by using a stress testing tool. See 4.7, “Stress testing tools” on page 157.

After the problem is captured, stop the HTTP server to end the trace, by entering:

```
ENDTCPSVR SERVER(*HTTP) HTTPSVR(MYSERVER)
```

In this command, replace *MYSERVER* with the name of your HTTP server.

After the trace ends, the output is written to the spooled file QZSRHTTPTTR. You can locate the spooled file by examining the spooled files for user QTMHHTTP or by examining the server job for your HTTP server, as illustrated in Figure 4-144.

```

                                Work with Job Spooled Files

Job:  MYSERVER      User:  QTMHHTTP      Number:  011480

Type options, press Enter.
  1=Send  2=Change  3=Hold  4=Delete  5=Display  6=Release  7=Messages
  8=Attributes      9=Work with printing status

Opt  File          Device or      Total   Current
      QZSRHTTPTTR  Queue      User Data  Status  Pages   Page  Copies
                        QPRINT      QSRV011476  HLD      21      1      1

```

Figure 4-144 HTTP tracing: Trace data is written to spool file QZSRHTTPTTR

If your HTTP server is already started and you cannot stop and restart it using the startup switches (for example, you have a production performance issue), you can still gather the same trace data by using the Trace TCP/IP Application (TRCTCPAPP) and Dump User Trace (DMPUSRTRC) commands. In Figure 4-145, we start a trace for the HTTP server by specifying *HTTP, *ON, the name of our server (MYSERVER), and the trace level of *ERROR. Notice that we also started an additional TCP/IP trace at the same time.

```

Trace TCP/IP Application (TRCTCPAPP)

Type choices, press Enter.

TCP/IP application . . . . . > *HTTP      *FTP, *SMTPSVR, *SMTPCLT...
Trace option setting . . . . .  *ON        *ON, *OFF, *END, *CHK
Maximum storage for trace . . .  *APP      1-16000, *APP
Trace full action . . . . .    *WRAP      *WRAP, *STOPTRC
Additional traces . . . . . > *TCPIP      *NONE, *CMNTRC, *TCPIP...
      + for more values
Trace program . . . . .        *NONE      Name, *NONE
Library . . . . .              Name, *LIBL, *CURLIB
HTTP server instance . . . . . > *MYSERVER Character value
Trace level . . . . .          *ERROR     *ERROR, *INFO, *VERBOSE
Configuration object . . . . .        Name
Type . . . . .                 *LIN, *NWI, *NWS

```

Figure 4-145 HTTP tracing: Using the TRCTCPAPP command

The DMPUSRTRC command is used to direct trace output to a database file while HTTP server jobs are active. The job name, number, and user profile for each one of your HTTP server jobs are required. Trace output is dumped to file QAP0ZDMP in QTEMP into a member called QP0Znnnnnn, where nnnnnn is the HTTP server job number you gave to the DMPUSRTRC command. Figure 4-146 illustrates the DMPUSRTRC command. The output from the command is found in member QP0Z011516 of the QAP0ZDMP file in QTEMP.

```

Dump User Trace (DMPUSRTRC)

Type choices, press Enter.

Job name . . . . . JOB          > MYSERVER
User . . . . .                > QTMHHTTP
Number . . . . .                > 011516
Trace record identifiers . . . . TRCRCID > *JOB

Output . . . . . OUTPUT        *FILE
Thread IDs to include . . . . . SLTTHD  *ALL
      + for more values
Thread IDs to exclude . . . . . OMTTHD   *NONE

```

Figure 4-146 HTTP tracing: Using the DMPUSRTRC command

Regardless of whether you use the browser-base HTTP administration interface, the STRTCPSVR command, or the TRCTCPAPP command, the HTTP server trace can be set to operate at three different levels: *error*, *information*, and *verbose*. Table 4-7 lists the startup options used on either the HTTP administration page or the STRTCPSVR command. It also shows the equivalent TRTCPAPP parameter together with the type of output you can expect to see.

Table 4-7 HTTP tracing: Startup options

| Startup switch | TRCTCPAPP parameter | Output |
|----------------|---------------------|--|
| -ve | *ERROR | Server startup only. Nothing else is recorded unless an error occurs |
| -vi | *INFO | Server startup and initialization, including directive processing, character conversion, and client request handling |
| -vv | *VERBOSE | All the above plus application programming interface (API) and module invocation, HTTP headers, error messages |

We recommend that you use -vv (*VERBOSE) when trying to resolve HTTP-related performance issues.

Tips:

- ▶ Always remember to stop the trace when it is no longer needed using:
 - a. Stop the server in the HTTP administration panel.
 - b. Enter the command:
ENDTCPSVR (*HTTP)
 - c. Enter this command:
TRCTCPAPP APP(*HTTP) SET(*OFF)
- ▶ When using the DMPUSRTRC command, remember that you can access only the content of the QTEMP library from your current session. It is discarded as soon as you sign off.

4.6.3 Collection Services

Beginning in V5R2, you can define your own performance collection categories within iSeries Collection Services. The HTTP Server (powered by Apache) uses this new feature to integrate performance data into Collection Services (see Figure 4-147).

As shown in Figure 4-147, the Standard plus protocol profile (the default used by Collection Services) automatically collects HTTP Server (powered by Apache) if Collection Services detects that the HTTP server is active on the system. As with all Collection Services collection object data, the new statistics are placed into the following files, using the iSeries Navigator Create performance database files function or the OS/400 CRTPFRTA command:

- ▶ QAPMHTTPPB: Contains the basic data for HTTP Server (powered by Apache)
- ▶ QAPMHTTPPD: Contains detailed data for HTTP Server (powered by Apache)

The Collection Services data can then be queried to analyze HTTP server performance and better understand the types of HTTP transactions that are being processed. In addition, V5R2 Performance Tools for iSeries (5722-PT1) has new sections in the System and Component Reports for HTTP statistics.

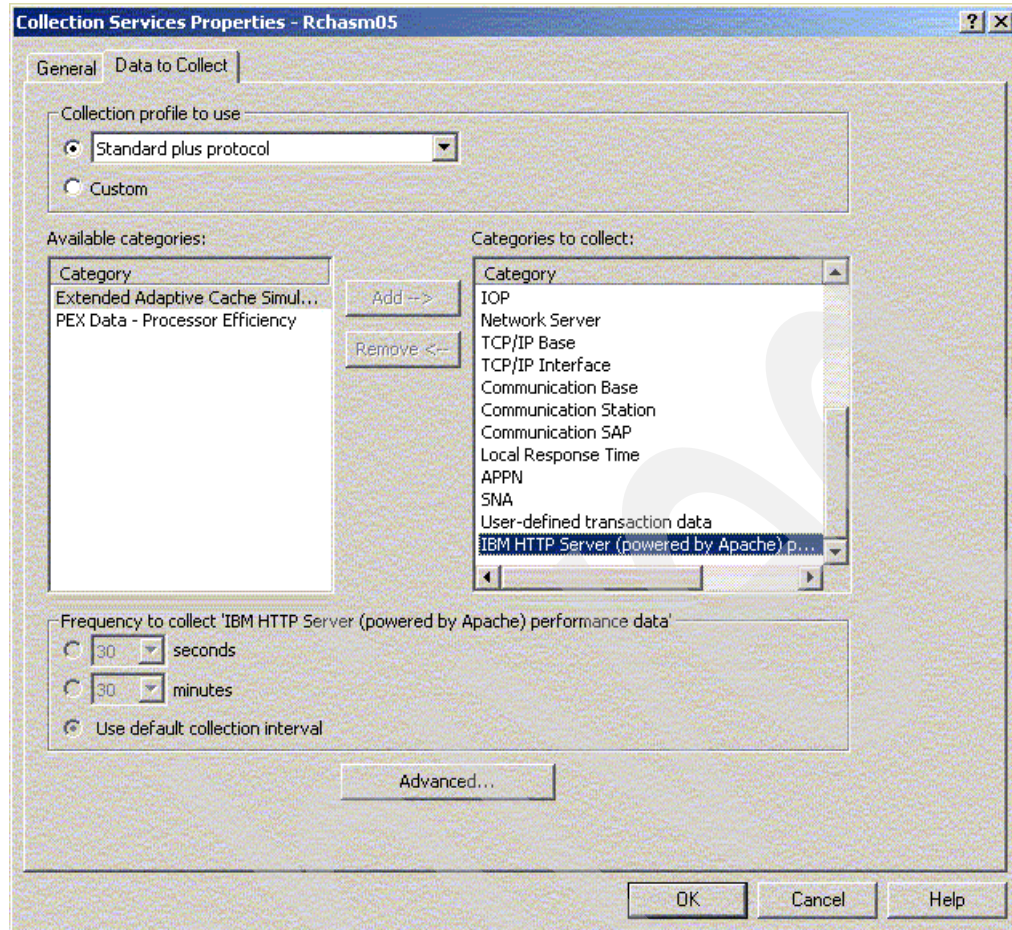


Figure 4-147 HTTP performance: HTTP integration with Collection Services

Performance data is collected per server job and the statistics are shown for each interval and request type within the interval. The types of data collected are broken down into the following ways:

- ▶ SR: Requests handled internally by the server itself; no program processing is necessary
- ▶ SL: Requests of all types received via SSL; reports activity that occurred over an SSL connection even though that activity is also reported with other applicable request types
- ▶ PX: Proxy requests
- ▶ CG: CGI requests
- ▶ WS: WebSphere requests
- ▶ JV: IBM Java Servlet Engine requests
- ▶ UM: Requests handled by user modules
- ▶ FS: Static requests handled by Fast Response Cache Accelerator (FRCA)
- ▶ FX: Requests proxied by FRCA

Starting Collection Services for the HTTP Server (powered by Apache)

We use iSeries Navigator to start and work with Collection Services. To start Collection Services, follow these steps:

1. Log on to your iSeries server using with iSeries Navigator.
2. Expand **your server** → **Configuration and Service**.
3. Right-click **Collection Services** and select **Start Collection**.
4. You see the Start Collection Services window (Figure 4-148). For the most part, you may accept all the defaults.

Take note of the library where you are storing collections since you need to know this later to find the files. Click **OK** to start Collection Services.

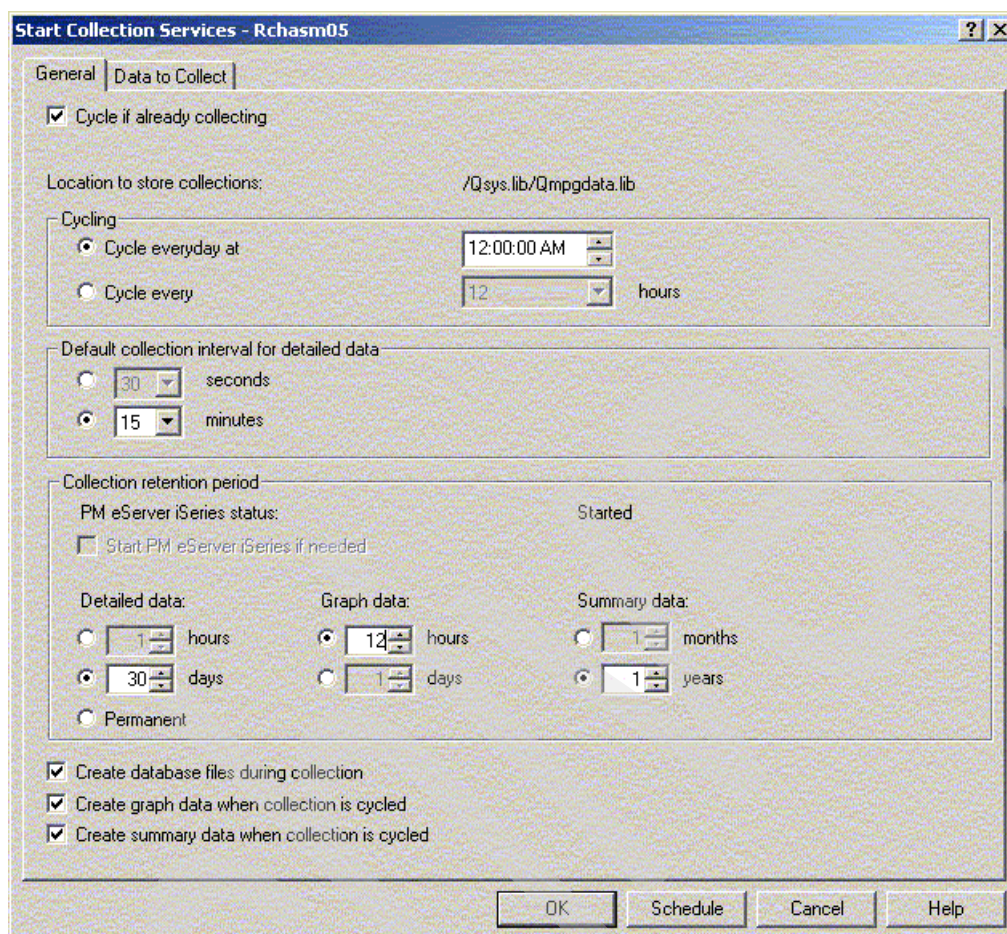


Figure 4-148 HTTP performance: Starting Collection Services

At this point, your HTTP Server (powered by Apache) server and related applications should be up and running. Collect the data for as long as you want.

When you are done collecting the data, stop Collection Services:

1. Right-click **Collection Services** and select **Stop Collection**.
2. In the Stop Collection Services panel, click **OK**.

Performance Tools reports

Performance Tools for iSeries were enhanced in OS/400 V5R2 to generate reports based on the HTTP performance data. The reports contain information about the transactions processed by the HTTP server jobs. Follow these steps to see the System and Component reports for the HTTP server:

1. From a 5250 session, enter the STRPFRT command.
2. Select option 3 (Print performance report).
3. Change the Library to QMPGDATA. Or, specify the library in which you saved the collection data. Press Enter to refresh the list of collections.

4. On the Print Performance Report - Sample data display, type option 1 (System report) to the left of the collection data member. Use the Date and Time columns to make sure you select the correct one.
5. On the Select Sections for Report display (Figure 4-149), press F6 to print the entire report. Or type 1 in front of any section to print that section.

```

                                Select Sections for Report

Member . . . . . : HTTP

Type options, press Enter. Press F6 to print entire report.
1=Select

Option      Section
            Workload
            Resource Utilization
            Resource Utilization Expansion
            Storage Pool Utilization
            Disk Utilization
            Communication Summary
            TCP/IP Summary
  
```

Figure 4-149 HTTP performance: Printing all sections of the system report

6. On the Select Categories for Report display (Figure 4-150), press F6 to print the entire report.

```

                                Select Categories for Report

Member . . . . . : HTTP

Type options, press Enter. Press F6 to print entire report.
1=Select

Option      Category
            Time interval
            Job
            User ID
            Subsystem
            Pool
            Communications line
            Control unit
            Functional area
  
```

Figure 4-150 HTTP performance: Printing all categories of the system report

7. On the Specify Report Options display, enter a meaningful report title. Press Enter to submit this work to the batch queue.
8. On the Print Performance Report - Sample data display, type option 2 (Component report).
9. Repeat these steps for the Component report.

When the batch jobs finish, you should have spooled files that contain the two reports. Figure 4-151 shows an example of the data that you may see in the Component report.

| HTTP Component Report | | | | | | | | | |
|-----------------------|-------------------|------------|------------------|------------------|----------|-------------|-----|-----------|------|
| Job Name | User Name/ Thread | Job Number | T y p e | P l a n | CPU Util | DB Cpb Util | Tns | Tns /Hour | Rsp |
| MYSERVER | QTMHHTTP | 011472 | B | 02 25 | .04 | .0 | 0 | 0 | .000 |
| MYSERVER | QTMHHTTP | 011473 | B | 02 25 | .02 | .0 | 0 | 0 | .000 |
| MYSERVER | QTMHHTTP | 011474 | B | 02 25 | .02 | .0 | 0 | 0 | .000 |
| MYSERVER | QTMHHTTP | 011475 | B | 02 25 | .04 | .0 | 0 | 0 | .000 |
| MYSERVER | QTMHHTTP | 011476 | B | 02 25 | .04 | .0 | 0 | 0 | .000 |
| MYSERVER | QTMHHTTP | 011479 | B | 02 25 | 11.26 | .0 | 0 | 0 | .000 |
| MYSERVER | QTMHHTTP | 011480 | B | 02 25 | 3.35 | .0 | 0 | 0 | .000 |

Figure 4-151 HTTP performance: Viewing the component report

To learn more about Performance Tools reports, consult the following resources:

- ▶ The iSeries Information Center has a starting point for performance-related topics that include the logging of information with iSeries Collection Services
<http://publib.boulder.ibm.com/iserics/v5r2/ic2924/info/rzahx/rzahxebushttp.htm>
- ▶ *AS/400 HTTP Server Performance and Capacity Planning*, SG24-5645

4.6.4 Communications trace

A communications trace is another a powerful tool for problem determination. It is useful for gathering information about the connection status, response time, among other things.

Here is a quick example of using a communications trace to analyze all the IP datagrams received and sent from the iSeries point of view. The communication trace commands that are used are:

- ▶ Start Communications Trace (STRCMNTRC)
- ▶ End Communications Trace (ENDCMNTRC)
- ▶ Print Communications Trace (PRTCMNTRC)
- ▶ Delete Communications Trace (DLTCMNTRC)

This example assumes that your configuration line objects name is ETHLINE. Enter the commands and perform the following actions in the order shown:

1. Enter the following command:
STRCMNTRC CFGOBJ(ETHLINE) CFGTYPE(*LIN) MAXSTG(256K) USRDTA(*MAX)
2. Start your HTTP Server (powered by Apache) and Web client. Test your Web application. Keep your work time to a minimum to lessen the amount of data collected.
3. Enter the following command:
ENDCMNTRC CFGOBJ(ETHLINE) CFGTYPE(*LIN)
4. Now enter this command:
PRTCMNTRC CFGOBJ(ETHLINE) CFGTYPE(*LIN) CODE(*ASCII) FMTBCD(*NO)
5. Enter the Work with Job (WRKJOB) command and select option 4 to work with spooled files. Figure 4-152 shows the type of output that you can see when viewing a communications trace.

| COMMUNICATIONS TRACE | | | | Title: Communications trace | | 07/20/04 |
|----------------------|----------------|--------|------------------|-----------------------------|------------------|---------------|
| Number | S/R | Length | Timer | Name | MAC Address | MAC Address |
| 25830 | R | 50 | 16:16:44.71861 | | 0002E55E1BE5 | 000F245663C25 |
| | Data | : | 450000283D214000 | 7B06CC45090A8825 | 09055C3511272AF8 | |
| | | | 5010FF3C28F30000 | 000000000000A1BB | BD30 | |
| 25831 | S | 1492 | 16:16:44.71930 | | 000F14565C20 | 0004AE5E1A15 |
| | Data | : | 450005D4B3884000 | 40068B3209055C35 | 090A88252B6110D5 | |
| | | | 501020009DDB0000 | 4142514141414439 | 3353586D42514141 | |
| | | | 4141414141414179 | 64785A7763334541 | 666741554141 | |

Figure 4-152 HTTP performance: Viewing the communications trace

6. Enter the following command:

```
DLTCMNTRC CFGOBJ(ETHLINE) CFGTYPE(*LIN)
```

For more information about how to perform a communications trace, refer to the IBM iSeries Support Knowledge Base document *TCP/IP Communications Trace Instructions* (document #23825849), which you can find at:

http://www-912.ibm.com/s_dir/slkbase.nsf/slkbase

4.7 Stress testing tools

Deploying applications that perform and scale in an acceptable manner is not an accidental occurrence. Producing high performance software requires that you include several rounds of stress testing during the development cycle. You can use one or more of the many open source or commercial stress testing tools to automate the execution of your stress tests.

The primary purpose of stress testing tools is to discover under what conditions your application's performance becomes unacceptable. You do this by changing the application inputs to place a heavier and heavier load on the application and measuring how performance changes with the variation in those inputs. This activity is also called *load testing*. However, load testing usually describes a specific type of stress testing, increasing the number of users to stress test your application.

The simplest way to stress test an application is to manually vary the inputs (such as the number of clients, size of requests, frequency of requests, mix of requests) and then chart how the performance varies. If you have many inputs, or a large range of values over which to vary those inputs, you may need an automated stress testing tool. Moreover, you want test automation to repeat test runs following environmental or application-specific changes after you uncover an issue. We recommend that you make only one change at a time in between test runs so the results are measurable.

If you are manually testing, it can be difficult to accurately reproduce an identical set of tests across multiple test executions. When it comes to having multiple users testing your application, it is almost impossible to run manual tests consistently. It can also be difficult to scale up the number of users testing the application.

Today, there is no generic, one-size-fits-all stress testing tool. Every application differs in what inputs it takes and how it executes them. Java and WebSphere-based Web applications generally receive requests from clients via the HTTP protocol. There are many stress testing tools that can simulate user activity over HTTP in a controlled and reproducible manner.

During our testing, we used both the IBM Rational® stress testing solution that includes the Test Manager and Robot tools as well as the open source stress testing tool OpenSTA. Available stress testing tools include such open source solutions as OpenSTA, Apache JMeter, The Grinder, and many others to a wide array of commercial options, such as the IBM Rational tools. “Stress testing tool resources” on page 164 provides links with more information about some of the available tools.

With so many stress testing tools available, how can you choose the one that is most appropriate for your application? Some of the points to consider when evaluating stress testing tools include:

- ▶ Client interaction
The stress testing tool must be able to handle the features and protocols that your application uses.
- ▶ Simulate multiple clients
This is the most basic functionality of a stress testing tool.
- ▶ Scripted execution with the ability to edit scripts
If you can't script the interaction between the client and the server, then you cannot handle anything except the most simple client requests. The ability to edit the scripts is essential. Minor changes shouldn't require you to go through the process of regenerating a script.
- ▶ Session support
If a stress testing tool does not support sessions or cookies, it isn't much of a stress testing tool and may not be able to stress test Java and WebSphere applications.
- ▶ Configurable numbers of users
The stress testing tool should let you specify how many simulated users are running each script or set of tasks, including allowing you to vary the number of simulated users over time. Many stress testing tools enable you to start with a small number of users and ramp up slowly to higher numbers of users.
- ▶ Reporting on success, errors, and failures
The tool you choose must have a defined way to identify a successful interaction, as well as failure and error conditions. An error may not receive any Web page back at all, where a failure may receive the wrong data back on the page.
- ▶ Page display and playback
A useful feature in many stress testing tools is the capability to inspect some of the pages that are being sent to the simulated users or to replay entire test scripts. You can then be confident that the stress test is functioning as you expect.
- ▶ Exporting test results
After running a stress test, you may want to analyze the test results using various tools that are external to the stress testing tool, including spreadsheets and custom analysis scripts. Most stress testing tools include extensive built-in analysis functions, but being able to export the data gives you more flexibility to analyze and catalog the data in arbitrary ways.
- ▶ Think time
Real-world users do not request one Web page immediately after another. There are generally delays between viewing one Web page and the next. The term *think time* is the standard way of expressing the addition of a delay into a test script to more realistically simulate user behavior. Many stress testing tools support randomly generated think times based on a statistical distribution.

- ▶ Variable data

Live users don't work with the same set of data on each interaction with your application. During a stress test, your simulated users should also be doing this. It is easier to make your simulated users appear to be working with varied data if the stress testing tool supports data input from lists, files, or a database.

- ▶ Script recording

Rather than writing scripts, it is much easier to manually run through a session with your browser and have that session recorded for later editing. Most stress testing tools include provisions for capturing manual interaction with your application.

- ▶ Analysis tools

Measuring performance is only half the story. The other and perhaps more important half of stress testing is analyzing the performance data. The type of analysis tools and degree of detailed analysis that you can perform depends directly on what analysis tools are supported by the tool you select. Therefore, evaluate this support in the tools that you are considering carefully.

- ▶ Load distribution

Your deployed application may need to support hundreds of concurrent users once it is in production. How can you simulate this level of traffic in a stress testing environment? A typical workstation running a stress testing tool will likely begin bottlenecking when approximately 200 virtual users are running. To simulate a greater number of users, you can distribute the stress testing load across multiple workstations. Many of the available stress testing tools support distribution of the load, which you will certainly want feature for large scale stress testing.

- ▶ Measuring server-side statistics

The basic stress testing tool measurement is client-based response times from client/server interactions. However, you may also want to gather other statistics, such as CPU utilization or page faulting rates. With this server-side data, you can then perform such useful tasks as view client response times in the context of server load and throughput statistics.

4.7.1 IBM Rational stress testing tools

The IBM Rational family of products includes several different tools, such as the Functional Tester for Java and Web. For our testing, we leveraged IBM Rational Robot for test script creation and IBM Rational Test Manager for stress test execution.

IBM Rational Test Manager provides a central console for test activity management, execution, and analysis. It supports everything from purely manual test approaches to various automated paradigms including unit testing, functional regression testing, and performance testing. IBM Rational Test Manager is meant to be accessed by all members of a project team. This ensures the high visibility of test coverage information, defect trends, and application readiness.

For our purposes, we simply created test scripts using IBM Rational Robot to exercise the various portions of the Trade 3 application, as shown in Figure 4-153. In this particular script, we did not simulate user think times. We created some test scripts with think times that were between 0 and 3 seconds.

```
push Http_control = HTTP_PARTIAL_OK | HTTP_CACHE_OK | HTTP_REDIRECT_OK;
push Timeout_scale = 200; /* Set timeouts to 200% of maximum response time */
push Think_def = "LR";
Min_tmout = 120000; /* Set minimum Timeout_val to 2 minutes */
push Timeout_val = Min_tmout;
push Think_avg = 0;

ITS0 = http_request ["PingSer~001"] "ITS0:80",
    HTTP_CONN_DIRECT,
    "GET /trade/servlet/PingServletWriter HTTP/1.1\r\n"
    "Accept: image/gif, image/x-bitmap, image/jpeg, image/pjpeg, applicat"
    "ion/x-shockwave-flash, application/vnd.ms-excel, application/vnd.ms-pow"
    "erpoint, application/msword, */*\r\n"
    "Accept-Language: en-us\r\n"
    "Accept-Encoding: gzip, deflate\r\n"
    "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR"
    " 1.1.4322)\r\n"
    "Host: ITS0\r\n"
    "Connection: Keep-Alive\r\n"
    "\r\n";

set Server_connection = ITS0;
```

Figure 4-153 Stress testing: Creating stress testing scripts using IBM Rational Robot

After creating multiple test scripts to simulate a mixed workload, we constructed several test suites (Figure 4-154) to execute one or more of the test scripts. The test suites allowed us to create a “repeatable” workload. We could measure performance for a given test suite execution and make a *single* adjustment (recommended) to our environment or application. Then we could run the same test suite a second time to analyze how the change we made affected performance.

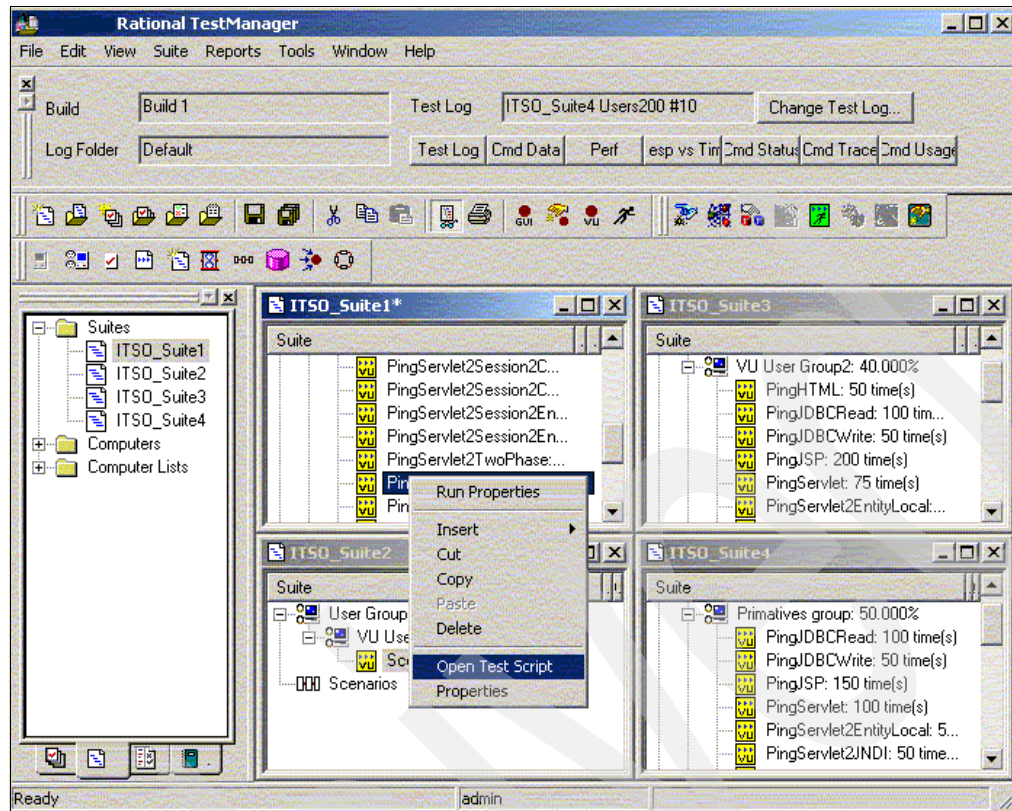


Figure 4-154 Stress testing: Executing test suites with IBM Rational Test Manager

IBM Rational tools support a broad array of functionality. We used the components and features that we needed to execute performance testing of our application. Some of the other major features of IBM Rational tools include:

- ▶ **Support for all test types**
Plan, manage and execute functional, performance, manual, integration, regression, configuration and component testing from the Test Manager interface.
- ▶ **Customizable test management**
APIs let users build support for new test inputs and test types.
- ▶ **Support for local and remote test execution**
Run tests on local or remote machines. Distributed, parallel test execution is limited only by the amount of system resources at your disposal.
- ▶ **Establish and manage traceability**
Requirements are linked to test cases, ensuring proper test coverage. In addition, suspicion analysis ensures that when requirements change, test cases traced to the requirement are automatically flagged as possible candidates for modification.
- ▶ **Detailed test evaluation**
An integrated log viewer constructs a log for each test run, including test status, environmental information, and such additional informational tags as runtime analysis information gathered by other IBM Rational tools, such as PurifyPlus™.
- ▶ **Generate meaningful reports**
The Rational Test Manager includes a series of predefined graphical and textual reports capturing the crucial aspects of application quality and test completeness. Special reports

exist for performance test analysis, including correlation reports combining response times with resource utilization metrics from remote machines.

- Integration with other IBM Rational tools

This includes RequisitePro®, ClearQuest®, and ClearCaseLT. For more information, see:

<http://www.ibm.com/software/rational>

4.7.2 OpenSTA

OpenSTA is an open source stress testing tool developed in C++ and released under the GNU GPL license. It is an HTTP stress test application with scenario creation and monitoring capabilities, which can be distributed across multiple computers to simulate large workloads.

After launching OpenSTA, we created scripts using the built-in Script Modeler (Figure 4-155) to simulate multiple user activities, including data access and think times. You can write and edit scripts within the Script Modeler or you can use an included feature to record and generate scripts automatically.

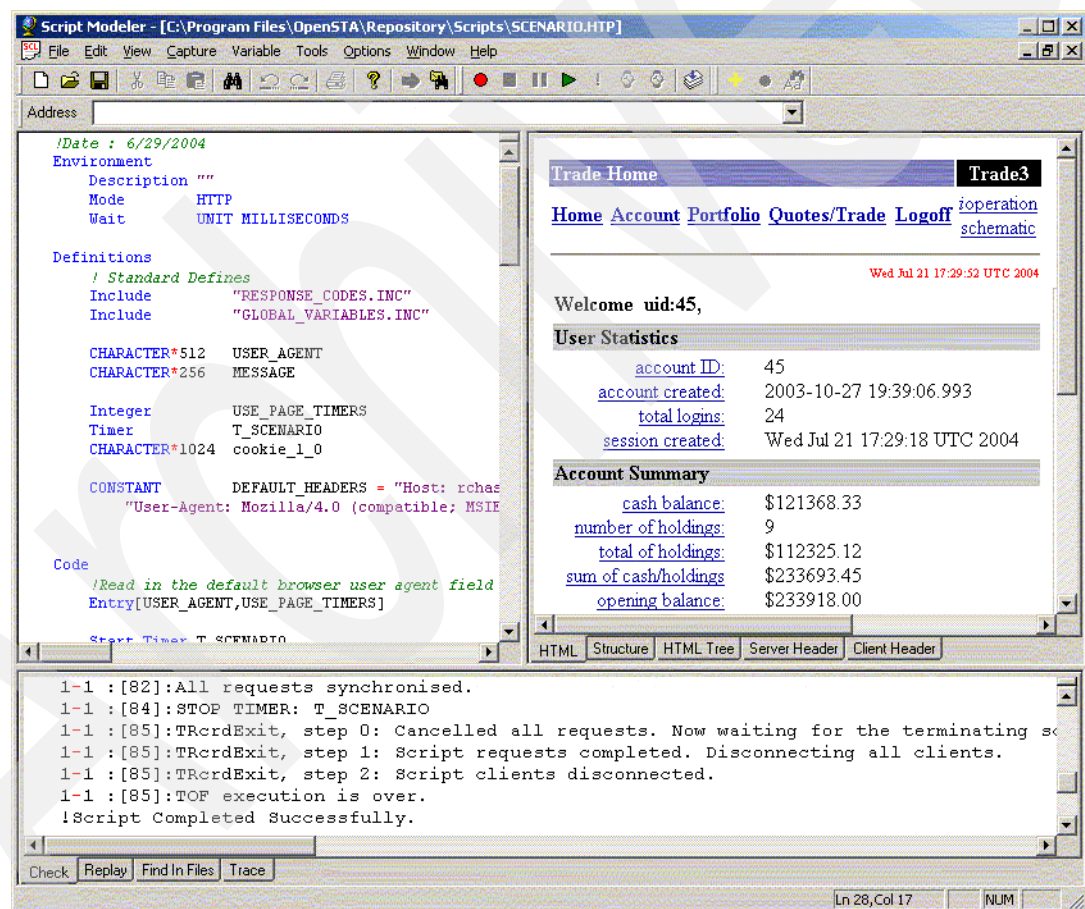


Figure 4-155 Stress testing: Using OpenSTA's Script Modeler to create scripts

The OpenSTA Commander is used to create, execute, and analyze test suites. Creating a test suite is simple. We needed to drag and drop various test scripts that we created into a test definition. Then, we specified which workstations would execute the stress test, the number of users to simulate, and the monitoring attributes we wanted. Figure 4-156 shows a test execution with summary user statistics.

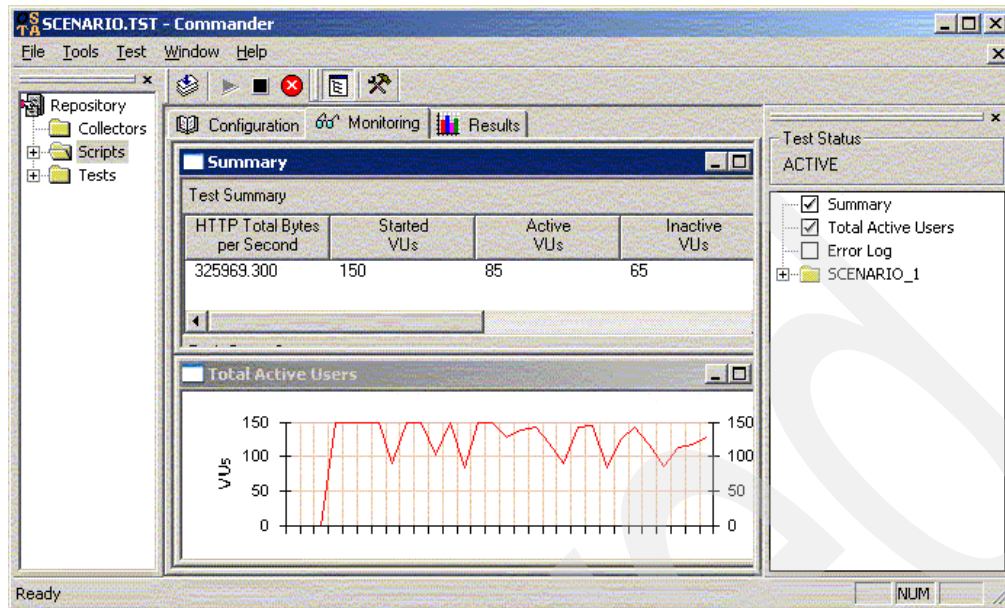


Figure 4-156 Stress testing: Running a test with OpenSTA Commander

During execution, you can monitor several parameters, such as CPU idle time, network traffic, disk usage, and HTTP errors.

After the stress test is completed, you can examine much of the information using the built-in analysis tools. System monitoring data (Simple Network Management Protocol (SNMP)) and test data are both captured. You can view all of the information in customizable graphs (Figure 4-157). And you can export data, such as a spreadsheet, for later review using other software.

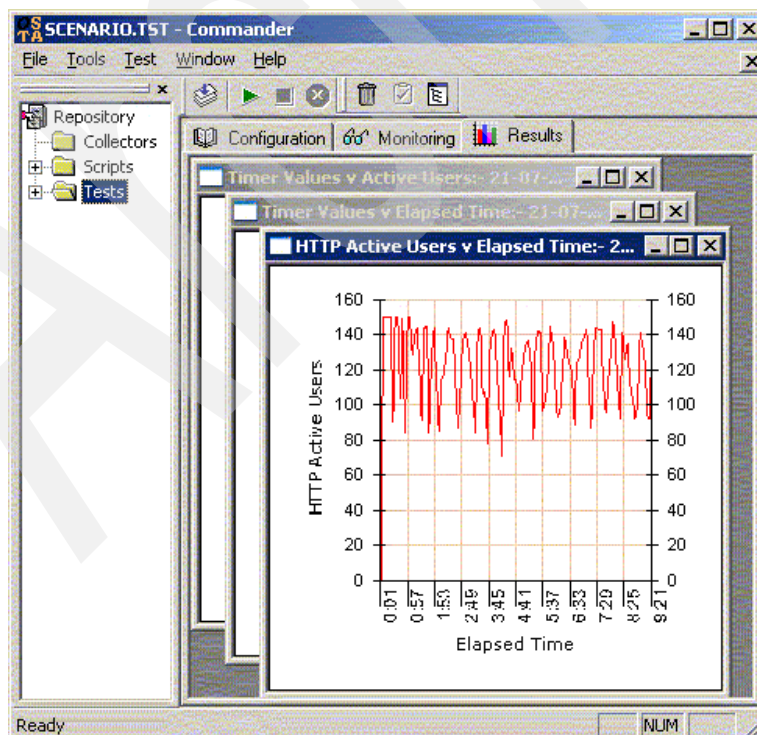


Figure 4-157 Stress testing: Reviewing results with OpenSTA analysis tools

Although OpenSTA is limited to the Windows platforms and HTTP-based testing only, it provides an easy-to-use user interface, good scripting and test facilities, and useful reporting that you can use to determine how to tune your applications to maximize performance. For more information, see the OpenSTA Web site:

<http://www.opensta.org>

Stress testing tool resources

For more information about stress testing, consult these sources:

- Software QA/Test Resource Center load and performance test tools

<http://www.softwareqatest.com/qatweb1.html#LOAD>

- Java Performance Tuning resources

<http://www.javaperformancetuning.com/resources.shtml>

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 reconfiguration 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 or 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 recommend that you use values and settings that will assist you in obtaining the best performance of the overall execution.

5.1 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 consider the network. Access to system services typically involves accessing databases. Traditionally, the iSeries server 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 input/output (I/O).

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

5.2 Verifying the state of Licensed Program Products

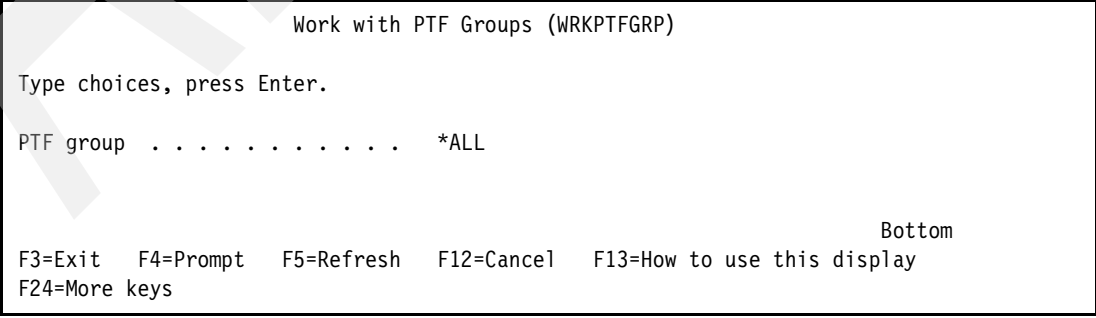
Before you even consider looking at operating system parameters, ensure that the operating system and licensed programs involved in your application are up to date. Errors found in licensed programs are corrected with program temporary fixes (PTFs) that are released either as cumulative packages (CUM PTF), group PTFs, or on an individual basis.

Performance fixes for Java are released frequently. 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

Group PTFs frequently contain PTFs that are not yet released in the cumulative PTF package. Use the Work with PTF Groups (WRKPTFGRP) command to determine the group PTF level on your system as shown in Figure 5-1.



```
Work with PTF Groups (WRKPTFGRP)

Type choices, press Enter.

PTF group . . . . . *ALL

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

Figure 5-1 Entering the WRKPTFGRP command

Figure 5-2 shows an example of the output created with the WRKPTFGRP command. The number following the Group PTF identifier states the PTF level.

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

| Work with PTF Groups | | | | System: | SYSTEMA |
|---|-----------|-------|-----------|---------|---------|
| Type options, press Enter. | | | | | |
| 4=Delete 5=Display 6=Print 9=Display related PTF groups | | | | | |
| Opt | PTF Group | Level | Status | | |
| | SF99502 | 13 | Installed | | |
| | SF99279 | 3 | Installed | | |
| | SF99277 | 3 | Installed | | |
| | SF99246 | 7 | Installed | | |
| | SF99245 | 8 | Installed | | |
| | SF99172 | 3 | Installed | | |
| | SF99169 | 17 | Installed | | |
| | SF99098 | 13 | Installed | | |
| | | | | Bottom | |
| F3=Exit F6=Print F11=Display descriptions F12=Cancel | | | | | |
| F22=Display entire field | | | | | |

Figure 5-2 Example of the WRKPTFGRP command output

Table 5-1 and Table 5-2 show the data areas to be verified.

Table 5-1 Group PTF numbers for various program products

| Program product | OS/400 V5R1 | OS/400 V5R2 | OS/400 V5R3 |
|-----------------|-------------|-------------|-------------|
| Java | SF99069 | SF99169 | SF99269 |
| HTTP Server | SF99156 | SF99098 | SF99099 |
| Database | SF99501 | SF99502 | SF99503 |

The group PTFs available for WebSphere Application Server are listed in Table 5-2.

Table 5-2 Group PTF identifiers for WebSphere versions

| WebSphere Application Server version | OS/400 V5R1 | OS/400 V5R2 | OS/400 V5R3 |
|--------------------------------------|-------------|---------------|---------------|
| Advanced Edition 3.5 | SF99147 | Not supported | Not supported |
| Standard Edition 3.5 | SF99146 | SF99146 | Not supported |
| Advanced Edition 4.0 | SF99241 | SF99148 | SF99289 |
| Advanced Edition 4.0, Single Server | SF99242 | SF99149 | SF99290 |
| ND 5.1 | SF99278 | SF99279 | SF99286 |
| BASE 5.1 | SF99276 | SF99277 | SF99285 |
| Express 5.1 | SF99273 | SF99274 | SF99275 |
| Express 5.0 | SF99270 | SF99271 | SF99272 |
| ND 5.0 | SF99244 | SF99246 | SF99288 |

| WebSphere Application Server version | OS/400 V5R1 | OS/400 V5R2 | OS/400 V5R3 |
|--------------------------------------|-------------|-------------|-------------|
| BASE 5.0 | SF99243 | SF99245 | SF99287 |

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

“Online resources” on page 373 lists Web sites that can help you obtain the latest information about updates. You may also contact your software service provider to check these levels.

5.3 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.

5.3.1 Manual tuning versus automatic tuning

First, you have to decide whether to tune your system manually or automatically. You may decide to use the automatic tuning to give you 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 182.

In both cases, you must check the settings of system values manually because the automatic tuner only changes the pool sizes and activity levels. The automatic tuner behavior is controlled with the setting of one system value, QPFRADJ. Setting this system value provides 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 you set the value 3, 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.

5.3.2 Semi-automatic system tuning

For example, during a low activity on your WebSphere or Java workload, the performance adjustment can, and probably will, remove memory from the pool in which those jobs are located if that memory is needed elsewhere. Then, if you have an increase in requests, it's possible that performance adjustment will not react fast enough to prevent paging. At best, this paging would cause a short performance issue for your Web site and certainly cause more critical issues if it is prolonged.

If you use the automatic tuner, but want to manually set up the size and activity level settings for a pool, then follow these steps:

1. Enter the Work with Shared Pools (WRKSHRPOOL) command.
2. Press F11 for the tuning data.
3. Set the minimum pool size equal to maximum pool size.

This tells the automatic tuner not to adjust this pool.

5.3.3 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 several system values that you should consider changing by using 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 in 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 even one spooled output file exists 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

First verify the settings of the allocation system values on your system. In some cases, this may be the only action that you need 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 WCBT entries created during the initial program load (IPL). You have two options to determine the required setting of this system value:

- ▶ Work with System Status (WRKSYSSTS) command
- ▶ Display Job Tables (DSPJOBTL) command

Enter the WRKSYSSTS command during a time of a high system activity and press F21 to set the assistance level to Advanced. You see a display similar to the example in Figure 5-3. 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.

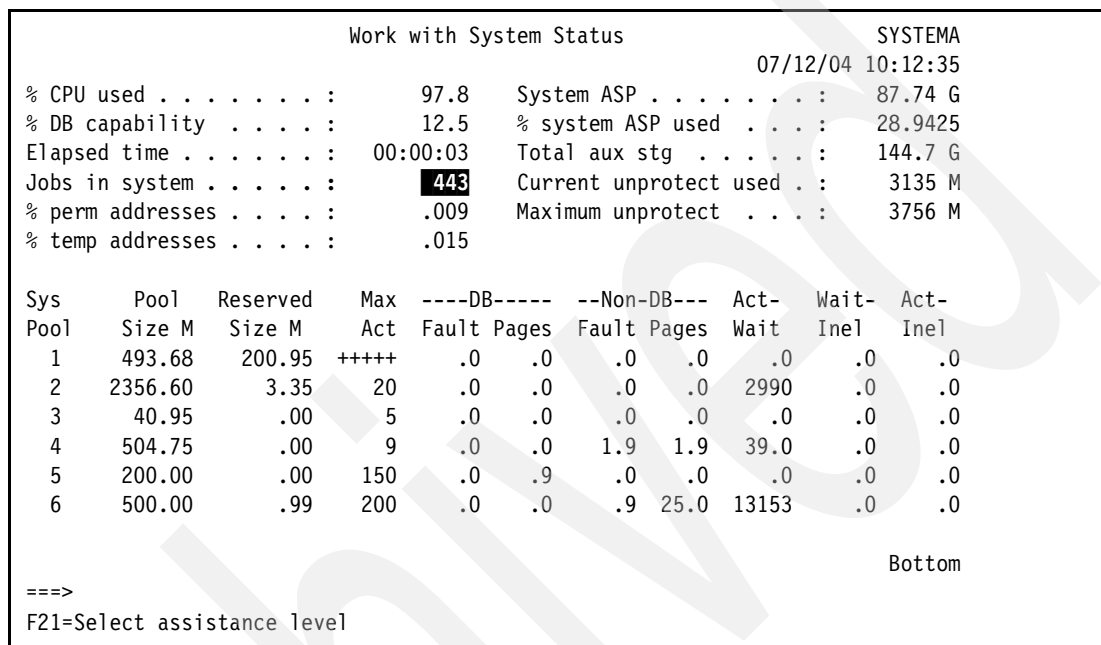


Figure 5-3 WRKSYSSTS with advanced assistance level setting

If you enter the DSPJOBTL command, 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 5-4 and Figure 5-5 show you what to expect when you enter the DSPJOBTL command. 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 the 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 5-4 with Figure 5-5.

This system value should be set 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 that happen 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 set the QTOTJOB too large rather 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.

```

                                Display Job Tables
                                SYSTEMA
                                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 5-4 DSPJOBTL showing the total number of WCBT entries

```

Display Job Tables
SYSTEMA
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 5-5 DSPJOBTL showing the WCBT entries currently in use

QMAXJOB (Maximum number of jobs)

This value specifies the maximum number of jobs that are allowed on the system. When the number of jobs reaches this maximum, you can no longer submit or start more jobs on the system. Use this system value to limit the storage used for job tables. The shipped value is 163520 and a change to this value takes effect immediately.

QADLTOTJ (Additional number of total jobs)

This value specifies how many WCBT entries will be created if the amount of jobs in the 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 recommend 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 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. You can do this by viewing the active jobs field on the active jobs display (using the Work with Active Jobs (WRKACTJOB) command) as shown in Figure 5-6. 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.

| Work with Active Jobs | | | | | | SYSTEMA |
|--|---------------|---------------|----------|--------------|---------------|-------------------|
| | | | | | | 10/12/01 10:13:42 |
| CPU %: | 99.1 | Elapsed time: | 00:05:43 | Active jobs: | 271 | |
| 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 |
| | QEJBAS51 | 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 5-6 Using WRKACTJOB to determine the QACTJOB system value setting

QADLACTJ (Additional number of active jobs)

This value 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 how system storage is being used. You must set the following values correctly:

- ▶ 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. 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 cannot accept more work because no activity level is available.

Use the WRKSYSSTS system command to adjust this setting. Increasing the value highlighted on the example shown in Figure 5-7 reduces or eliminates the thread transitions into the ineligible state.

Setting QBASACTLVL 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 | | | | | | | | SYSTEMA | |
|-----------------------------|---------|----------|----------------------------------|------------|-------------|-------|-------|----------|----------|
| | | | | | | | | 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 |
| Bottom | | | | | | | | | |
| ==> | | | | | | | | | |
| F21=Select assistance level | | | | | | | | | |

Figure 5-7 Using WRKSYSSTS to change the QBASACTLVL

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

If you are using a system with a mixed load of traditional high-level language (HLL) code and Java code, you should create a separate pool for the Java work. You can find an example of this in 5.3.4, “Creating a separate memory pool for a subsystem” on page 174.

QBASPOOL (Base storage pool minimum size)

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

QMAXACTLVL (Maximum activity level of system)

This value 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 was 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 that you use the setting *NOMAX. A change to this system value takes effect immediately.

QMCHPOOL (Machine storage pool size)

This value is perhaps the most important system value because it specifies the amount of main storage that is reserved for both the operating system programs and Licensed Internal Code (LIC). Enter the WRKSYSSTS command as shown in Figure 5-3 on page 170. 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 not where it is needed.

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

5.3.4 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 10 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

This memory pool 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 have 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 QEJBAS51, the subsystem in 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 Expert Cache” on page 183 for details.

Directing the QEJBAS51 jobs to a pool of their own

To direct jobs in a subsystem to a memory pool of their own, use these steps:

1. Enter the WRKSHRPOOL command to create a memory pool to be used for the subsystem jobs. In the Work with Shared Pools display (Figure 5-8, select a pool and set the initial size and activity level. 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 only initial values, but we refine them later by using the WRKSYSSTS command.

```

Work with Shared Pools
System: SYSTEMA

Main storage size (M) . . : 4096.00

Type changes (if allowed), press Enter.

  Defined      Max      Allocated      Pool      -Paging Option--
Pool  Size (M)  Active  Size (M)      ID  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
*SHRPOOL2 800.00    200  200.00      6  *CALC   *FIXED
*SHRPOOL3  .00      0    .00          7  *FIXED
*SHRPOOL4  .00      0    .00          8  *FIXED
*SHRPOOL5  .00      0    .00          9  *FIXED
*SHRPOOL6  .00      0    .00         10  *FIXED

More...

Command
===>
F3=Exit  F4=Prompt  F5=Refresh  F9=Retrieve  F11=Display tuning data
F12=Cancel

```

Figure 5-8 Creating a separate memory pool for QEJBAS51

2. Enter the CHGSBSD command to tell the subsystem that it has a new memory pool available:

```
CHGSBSD SBSD(QEJBAS51/QEJBAS51) POOLS((2 *SHRPOOLX))
```

Figure 5-9 shows you an example of using the Change Subsystem Description command. By default, subsystem pool 1 is where the subsystem monitor will run. We recommend that you use the *BASE pool for this pool identifier and use 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 memory in the same pool.

Change Subsystem Description (CHGSBSD)

Type choices, press Enter.

| | | | |
|---------------------------------|--------------|--------------------|--|
| Subsystem description | SBSD | > QEJBAS51 | |
| 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 5-9 CHGSBSD QEJBAS51 display

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

```
CHGRTGE SBSDB(QEJBAS51/QEJBAS51) SEQNBR(9999) POOLID(2)
```

As shown in Figure 5-10, direct the jobs to the pool created in the previous step. With the CHGRTGE command, we specify that all the jobs in the QEJBAS51 that use the routing entry with sequence number 9999 will go to the subsystems memory pool 2.

Change Routing Entry (CHGRTGE)

Type choices, press Enter.

| | | |
|---|-----------------|-----------------------|
| Subsystem description | QEJBAS51 | Name |
| Library | QEJBAS51 | 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, *SBSDB |
| 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 5-10 CHGRTGE to direct the jobs to the pool

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

The CHGRTGE command does not affect the jobs that are 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.

5.3.5 Setting the memory pool sizes and activity levels manually

You can perform manual adjustment using both the WRKSHRPOOL and WRKSYSSTS commands. This section shows you how to use the WRKSYSSTS command to tune the system. All of the windows were captured using the *advanced* assistance level.

Determining the memory pool size via 5250 screen

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 (known as a *page fault*).

As data is transferred into a memory pool, other data is displaced. If the data is changed, it is automatically recorded on disk (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 QEJBAS51 jobs are running.
 - a. Enter the Work with Subsystems (WRKSBS) command.
 - b. You see the display shown in Figure 5-11. Press the F14 key or enter the WRKSYSSTS command.

```

Work with Subsystems
System: SYSTEMA

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

  Total
Opt Subsystem Storage (M) -----Subsystem Pools-----
    QBATCH      .00      1  2  3  4  5  6  7  8  9 10
    QCMN         .00      1  2  3  4  5  6  7  8  9 10
    QCTL         .00      1  2  3  4  5  6  7  8  9 10
    QEJBAS51     .00      1  2  3  4  5  6  7  8  9 10
    QHTTSPVR     .00      1  2  3  4  5  6  7  8  9 10
    QINTER       .00      1  2  3  4  5  6  7  8  9 10
    QSERVER      .00      1  2  3  4  5  6  7  8  9 10
    QSPL         .00      1  2  3  4  5  6  7  8  9 10
    QSYSWRK      .00      1  2  3  4  5  6  7  8  9 10
    QUSRWRK      .00      1  2  3  4  5  6  7  8  9 10

Bottom

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

```

Figure 5-11 Using the WRKSBS command to relate the memory pools with WRKSYSSTS

2. You see the display shown in Figure 5-12. Note that pool 2 for subsystem QEJBAS51 is *not* pool 2 on the WRKSYSSTS display. The memory pools are allocated in the order in which the subsystems are started. In this case, pool 6 on the WRKSYSSTS display is pool 2 for the QEJBAS51.
3. Display the faulting data that represents the rate that data is moved between main storage and disk. This is shown by the two columns in the middle of the screen. The columns are labeled DB Faults and Pages and Non-DB Faults and Pages.
Every row represents a separate memory pool. In our example, we concentrate on pool 6.
4. Pay attention to the page fault rate (DB versus Non-DB). DB faults and pages indicate how the data is moved. Non-DB faults and pages represent the program code, job work areas, variables, and so on.
Increase the pool size to reduce faulting or reduce the amount of activity levels which is described next. What is a good number to strive for? See "Tuning main storage with the WRKSYSSTS command" on page 182 for guidelines.

| Work with System Status | | | | | | | | | | SYSTEMA | |
|------------------------------|---------|----------|----------|------------------------------------|-------|------------|-------|-------|-------|----------|----------|
| | | | | | | | | | | 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 5-12 Using WRKSYSSTS to determine the size of a memory pool

Determining the initial pool size via Job Watcher

The Job Watcher provides a quick way to determine the minimum size of a pool where the WebSphere or Java jobs are running in. Perform the following steps:

1. Start a Job Watcher collection (see “Starting a Job Watcher collection” on page 42).
2. Select the Summary graphs.
3. Open the Java Heap sizes graph. Observe the size of the heap as shown in Figure 5-13.
4. Multiply the minimum occurring heap size by 2 and set that to the pool size using the WRKSYSSTS command.

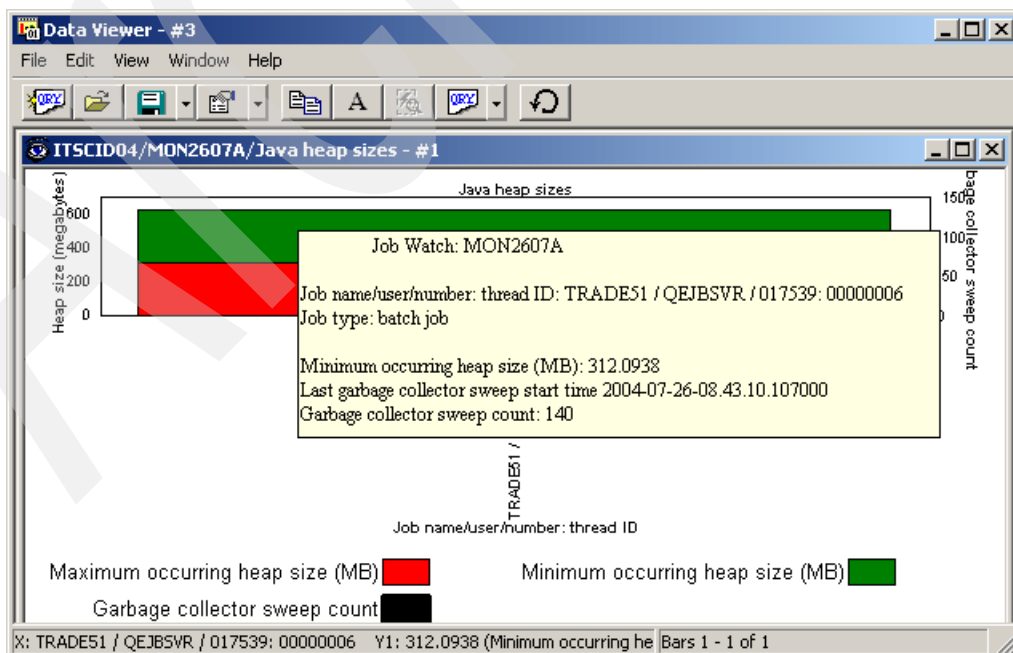


Figure 5-13 Using a Job Watcher Collection to determine the initial pool size

Determining the initial activity level setting for a pool via WRKACTJOB

Follow these steps to determine the initial setting for activity levels in a pool:

1. Enter the WRKACTJOB command.
2. On the Work with Active Jobs display (Figure 5-12), scroll down to the subsystem you work with. In this example, we use the QEJBAS51. Press F11 twice.

| Work with Active Jobs | | | | | | SYSTEMA |
|-------------------------------|---------------|---------------|----------|--------------|---------------|-------------------|
| | | | | | | 07/26/04 12:07:39 |
| CPU %: | 99.9 | Elapsed time: | 00:04:55 | Active jobs: | 501 | |
| Opt | Subsystem/Job | User | Type | CPU % | Function | Status |
| | QBATCH | QSYS | SBS | .0 | | DEQW |
| | QCMN | QSYS | SBS | .0 | | DEQW |
| | QCTL | QSYS | SBS | .0 | | DEQW |
| | DSP01 | ITSCID04 | INT | .0 | CMD-WRKSYSSTS | DSPW |
| | QSYSSCD | QPGMR | BCH | .0 | PGM-QEZSCNEP | EVTW |
| | QEJBAS51 | QSYS | SBS | .0 | | DEQW |
| | QEJBMQLSR | QEJBVR | BCI | .0 | PGM-RUNMQLSR | TIMW |
| | TRADE51 | QEJBVR | ASJ | 70.7 | * -COMMIT | JVAW |
| | QHTTPSVR | QSYS | SBS | .0 | | DEQW |
| | ADMIN | QTMHHTTP | BCH | .0 | PGM-QZHBHTTP | SIGW |
| | ADMIN | QTMHHTTP | BCI | .0 | PGM-QZSRLOG | SIGW |
| | ADMIN | QTMHHTTP | BCI | .0 | PGM-QZSRHTTP | SIGW |
| | ADMIN | QTMHHTTP | BCI | .0 | PGM-QZSRCGI | TIMW |
| | TRADE51 | QTMHHTTP | BCH | .0 | PGM-QZHBHTTP | SIGW |
| | TRADE51 | QTMHHTTP | BCI | .0 | PGM-QZSRLOG | SIGW |
| | TRADE51 | QTMHHTTP | BCI | .0 | PGM-QZSRLOG | SIGW |
| | | | | | | More... |
| ====> | | | | | | |
| F21=Display instructions/keys | | | | | | |

Figure 5-14 Using the WRKACTJOB for the initial activity level setting

3. You see the thread data as shown in Figure 5-15. Add the number of all the threads running in the pool. Set that number as the initial activity level via WRKSYSSTS or WRKSHRPOOL command.

| Work with Active Jobs | | | | | | SYSTEMA |
|-------------------------------|---------------|---------------|----------|--------------|-------|-------------------|
| | | | | | | 07/26/04 17:12:62 |
| CPU %: | 99.9 | Elapsed time: | 00:04:55 | Active jobs: | 501 | |
| Opt | Subsystem/Job | User | Number | Type | CPU % | Threads |
| | QBATCH | QSYS | 015170 | SBS | .0 | 1 |
| | QCMN | QSYS | 015172 | SBS | .0 | 1 |
| | QCTL | QSYS | 015145 | SBS | .0 | 1 |
| | DSP01 | ITSCID04 | 017568 | INT | .0 | 1 |
| | QSYSSCD | QPGMR | 015165 | BCH | .0 | 1 |
| | QEJBAS51 | QSYS | 017602 | SBS | .0 | 1 |
| | QEJBMQLSR | QEJBSVR | 017620 | BCI | .0 | 3 |
| | TRADE51 | QEJBSVR | 017603 | ASJ | 70.7 | 347 |
| | QHTTPSVR | QSYS | 016545 | SBS | .0 | 1 |
| | ADMIN | QTMHHTTP | 016546 | BCH | .0 | 1 |
| | ADMIN | QTMHHTTP | 016547 | BCI | .0 | 1 |
| | ADMIN | QTMHHTTP | 016548 | BCI | .0 | 30 |
| | ADMIN | QTMHHTTP | 016553 | BCI | .0 | 1 |
| | TRADE51 | QTMHHTTP | 017078 | BCH | .0 | 1 |
| | TRADE51 | QTMHHTTP | 017079 | BCI | .0 | 1 |
| | TRADE51 | QTMHHTTP | 017080 | BCI | .0 | 1 |
| | | | | | | More... |
| ====> | | | | | | |
| F21=Display instructions/keys | | | | | | |

Figure 5-15 WRKACTJOB showing the thread data

Determining the activity level setting for a pool via WRKSYSSTS

To determine the activity level for setting a pool, view the transition data that is highlighted in Figure 5-16. Pay attention to the three right most columns. Every row represents a separate memory pool. It is normal for threads to go from *active* status to *wait* status.

The rate of change is important here.

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

| Work with System Status | | | | | | | | SYSTEMA | | | |
|------------------------------|---------|----------|-------|-----------|-------------------------------|------------|-------|----------|----------|------|--|
| | | | | | | | | 07/24/01 | 17:12:62 | | |
| % 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 5-16 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.

Tuning main storage with the WRKSYSSTS command

The 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 the 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 low page fault rates (<0.4), 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 must 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 W->I, increase the MAXACT by a value of 5 to 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. Wait at least one minute between refreshes.
5. We recommend that you keep the rate of faulting as low as possible for the pools that the WebSphere and Java jobs are running. We have noticed that a faulting rate of over 100 pages per second can seriously impact the garbage collector's ability to manage the heap and collect the unneeded objects. This causes even more paging and a serious performance impact on the WebSphere or Java jobs running within the pool.
6. Omit a pool reserved for WebSphere or Java jobs from being adjusted by the automatic tuner as explained in 5.3.2, "Semi-automatic system tuning" on page 168.
7. For the user pools, the fault rate alone is not necessarily a measure of good or poor performance. It only indicates that data, program objects, or both are moved between the disk and the main storage. Response time and throughput are usually the actual metrics of measuring the performance. Therefore, 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.

8. If you cannot achieve a good performance by balancing the memory between the pools, consider adding more memory or even a more powerful processor to the system.

Enabling 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.

5.3.6 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 perform Java Database Connectivity (JDBC) access to DB2 Universal Database (UDB) for iSeries. The native JDBC functions are carried out by the QSQSRVR jobs running on QSYSWRK, where IBM Toolbox uses the QZDASOINT jobs that are running on QUSRWRK subsystem.

Make sure that these server jobs have the resources that they need when they need them. Otherwise 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 you may use the same approach with the QZDASOINIT jobs.

This section explains how to improve a server job's 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 recommend 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/>

When you reach this site, 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, 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.

Compare the values shown in Figure 5-17 with the current setting for the prestarted job values shown in Figure 5-20 on page 186. Then you can decide that the default values are not sufficient enough.

| Display Active Prestart Jobs | | | | SYSTEMA |
|----------------------------------|---------|------------------------|------------|-------------------|
| | | | | 20/07/04 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 5-17 Display Active Prestart Jobs

Determining the number of prestart jobs you start

This is shown in the subsystem descriptions prestart job entries part.

1. You enter the Display Subsystem Description (DSPSBSD) command as shown here:
 DSPSBSD SBSD(QSYSWRK)

2. You see the Display Subsystem Description screen (Figure 5-18). Choose option 10 (Prestart job entries).

```

                                Display Subsystem Description
                                System:  SYSTEMA
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 5-18 Display Subsystem Description

3. In the Display Prestart Job Entries screen (Figure 5-19), choose option 5 (Display details).

```

                                Display Prestart Job Entries
                                System:  SYSTEMA
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
    5  QSQSRVR      QSYS        QUSER
    QSRRATBL      QSYS        QUSER
    QTMSRVR       QTCP        QTCP
    QTMSCLCP      QTCP        QTCP
    QTMSRCP       QTCP        QTCP
    Q5BWHSRV      QSYS        QUSER

Bottom

F3=Exit  F9=Display all detailed descriptions  F12=Cancel
  
```

Figure 5-19 Display Prestart Job Entries

Now you see the prestart job entry QSQSRVR as shown on Figure 5-20. On this screen, you see that the operating system starts five QSQSRVR prestart jobs when subsystem QSYSWRK is started. When you compare this display with Figure 5-17 on page 184, you see that 38 prestart jobs were started during the time that the subsystem was active.

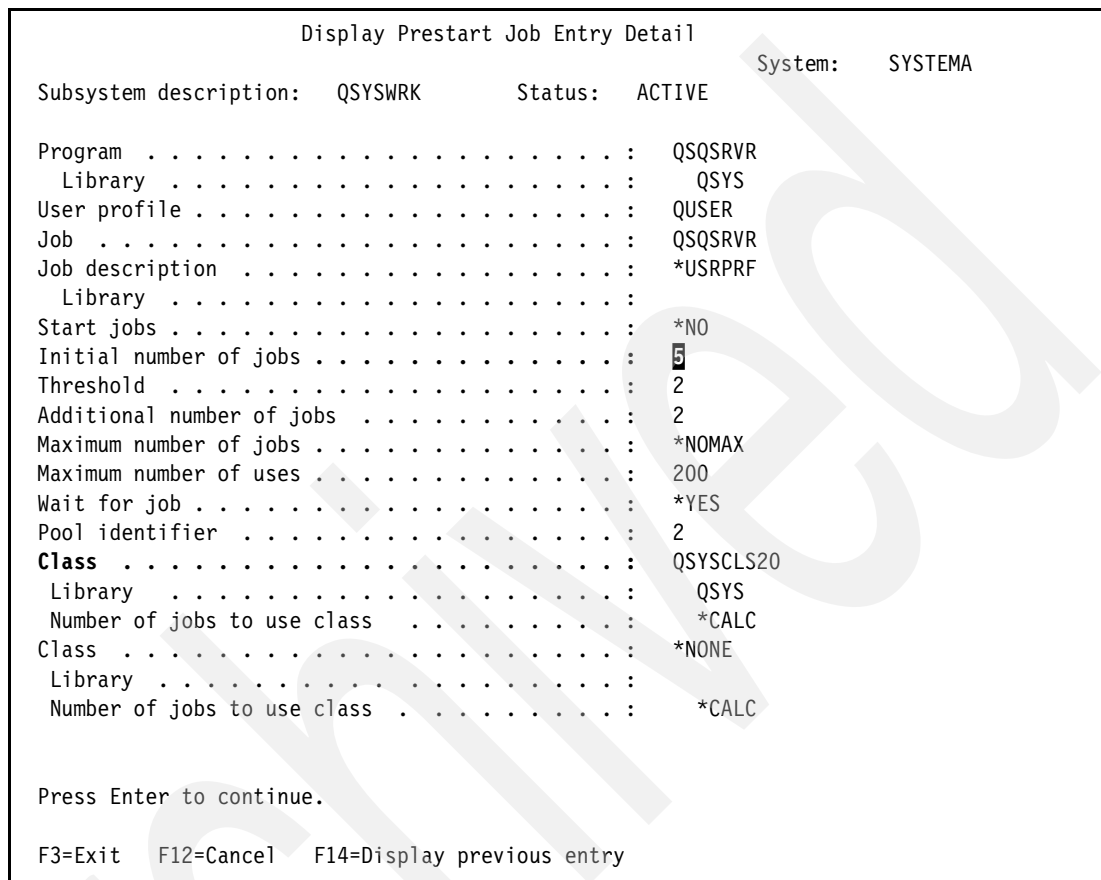


Figure 5-20 Display Prestart Job Entry Detail

We recommend that you start the required amount of the prestart jobs when you start 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 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 number of the prestart jobs required is based on the application needs. If you have an application that uses a large number of database connects and disconnects, you need a lot of prestart jobs. If your application uses a relatively small number of database connects and disconnects, you 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. Examine the output shown in Figure 5-17 on page 184 to learn the number of prestart jobs you have currently running. Make sure that you start at least that number of jobs when starting your subsystem.

Specifying the number of prestart jobs started

You specify the number of prestart jobs that are started by using the CHGPJE command, as shown in Figure 5-21.

```
Change Prestart Job Entry (CHGPJE)

Type choices, press Enter.

Subsystem description . . . . . SBSDB      > 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 . . . . . JOBD            *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 5-21 Change Prestart Job Entry command

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 5.3.4, “Creating a separate memory pool for a subsystem” on page 174, for details about 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 QSQSRVR jobs to that pool, 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 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.

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.
2. You see the WRKSHRPOOL display as shown in Figure 5-22. Specify the memory pool size, activity level setting, and paging option for SHRPOOLX.

| Work with Shared Pools | | | | | | |
|--|---------------------|---------------|-----------------------|------------|-----------------------------|---------|
| | | | | | System: | SYSTEMA |
| 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 | 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 5-22 WRKSHRPOOL display

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

```
CHGSBSD SBSD(QSYSWRK) POOLS((2 *SHRPOOLX))
```

4. Enter 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:

```
ENDPJ SBSD(QSYSWRK) PGM(QSQSRVR)
STRPJ SBSD (QSYSWRK) PGM(QSQSRVR)
```

5.4 Java system environment

If your iSeries server is running only a Java workload, 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 must understand how OS/400 determines the RUNPTY for each thread.

5.4.1 Java thread run 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

Here, the interactive default BCI job priority is 14. The BCI job priority for WebSphere comes from WebSphere Application Server configuration and by default is equal to 20 (see Figure 5-23).

| General Properties | | |
|--------------------|-----|--|
| Process Priority | 20 | <i>i</i> Specifies the operating system priority for the process. Only root users can change this value. |
| UMASK | 000 | <i>i</i> Specifies the user mask that the process runs under (the file-mode permission mask). |
| Run As User | | <i>i</i> Specifies the user that the process runs as. |

Figure 5-23 Process priority setting

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

```
CHGCLS QEJBAS51/QEJBCLS RUNPTY(...)
```

5.4.2 Java thread 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, enter:

```
CHGCLS QEJBAS51/QEJBCLS TIMESLICE(...)
```

5.5 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 Display Network Attributes (DSPNETA) command. To disable AnyNet support (if it is not used by other applications), use:

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

5.6 Example of an OS/400 tuning process

In this exercise, our workload consisted of 200 users, each of whom created one hundred transactions. In the beginning, the system required almost *an hour* to process this workload. Also, some transactions did not complete because of time outs. After we tuned the system, as described in the following sections, we ran the workload again and achieved a final processing time of only 9 minutes and 30 seconds.

Check the settings of the system values

First we verified that the system values were correctly set up, as discussed in 5.3.3, “Verifying the settings of system values” on page 169. In this case, these settings were correct.

Check the CPU utilization

Next, we looked at the overall system performance, using the Work with System Status (WRKSYSSTS) command as shown in Figure 5-24.

| Work with System Status | | | | | | | | | |
|------------------------------|---------|----------|-------|------------------------------------|-------------|-------|-------|----------|-----------|
| | | | | | | | | ASM05 | |
| | | | | | | | | 07/20/04 | 13:02:53 |
| % CPU used : | | 45.1 | | System ASP : | | | | 157.9 G | |
| % DB capability : | | 25.3 | | % system ASP used : | | | | 28.5342 | |
| Elapsed time : | | 00:08:02 | | Total aux stg : | | | | 157.9 G | |
| Jobs in system : | | 2610 | | Current unprotect used : | | | | 2482 M | |
| % perm addresses : | | .007 | | Maximum unprotect : | | | | 2482 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 | 535.66 | 226.69 | +++++ | .0 | .0 | 7.5 | 7.5 | 85.3 | .0 .0 |
| 2 | 7257.48 | .32 | 171 | .0 | .0 | .9 | 21.8 | 28.4 | .0 .0 |
| 4 | 1528.75 | .00 | 10 | .0 | .0 | 1.4 | 1.4 | 28.4 | .0 .0 |
| 5 | .25 | .00 | 1 | .0 | .0 | .0 | .0 | .0 | .0 .0 |
| 6 | 750.00 | .31 | 130 | .4 | 3.3 | 301.4 | 3129 | 1080 | .0 .0 |
| | | | | | | | | | |
| Bottom | | | | | | | | | |
| ==> | | | | | | | | | |
| F21=Select assistance level | | | | | | | | | |

Figure 5-24 WRKSYSSTS display for the tuning exercise

We checked the CPU utilization, which was well below the critical line at only 45% in a period of eight minutes. See the % CPU used and Elapsed time fields on the left in Figure 5-24. When the CPU is not the resource that is causing the bottleneck, you must continue the search and look at the main storage usage.

Check the memory utilization

There is no direct way to measure how much storage is in use at any given time. We can only measure that indirectly by observing the amount of paging and faulting per memory pool. A pool that is too small for the jobs running in it shows a large number of faulting, as compared to a pool that is sized correctly.

Remember that there usually is an amount of paging and faulting of data in the database, which is shown in the Database Fault and Pages columns in the WRKSYSSTS display (Figure 5-24). The amount of faulting depends on the workload and the design of the database and the way the application is written. Since the data is stored on the disk, it has to be brought into the main storage before it can be processed in the CPU.

If the size of a main storage pool is too small for the workload, the system may suffer from *trashing*, where the data is paged out from the main storage before it is processed in the CPU. This kind of situation is usually shown as a heavily fluctuating CPU utilization percentage.

Check the activity level setting in the pool

In this example, the pool that required our attention was pool number 6. The activity level setting (see the Max Act column in Figure 5-24) seemed correct since there were no thread state transitions shown in the three right-most columns.

You can find the correct setting for the activity level of a memory pool by using these steps:

1. Enter the WRKACTJOB command.
2. Press F11 twice to display the thread data.
3. Add the number of all the threads that are being run in the memory pool, as well as a 15% safety factor to determine the setting of the activity level.
4. Set that value for the pool by using either the WRKSYSSTS or WRKSHRPOOL command, or in the case of a private pool, by changing the subsystem description

However, the rate of both non-database faults and pages was way too high when compared with the rates of database faults and rates in the same pool. When combining the data where the CPU utilization was relatively low, with a high rate of faulting and a poor performance, we could safely assume that the CPU was mostly used for faulting instead of processing the data.

Check the disk utilization

You verify disk utilization by looking at the Work with Disk Status (WRKDSKSTS) display as shown in Figure 5-25. Usually, the disk utilization percentages (the right most column on the display) should not exceed 40% in a period longer than a minute. In this example, the disk units were used well above that guideline, as even the least busy units have 66% utilization.

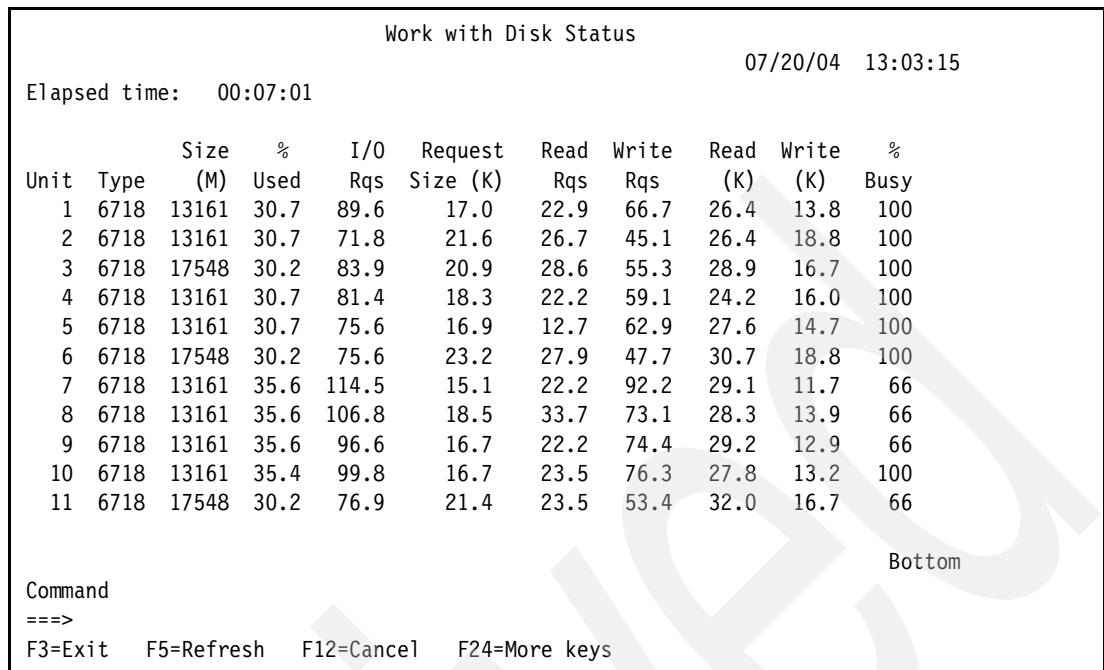


Figure 5-25 Using the WRKDSKSTS for problem determination

The easiest way to confirm the paging as the bottleneck is to use the iSeries Navigator System Monitor that provides the graph shown in Figure 5-26.

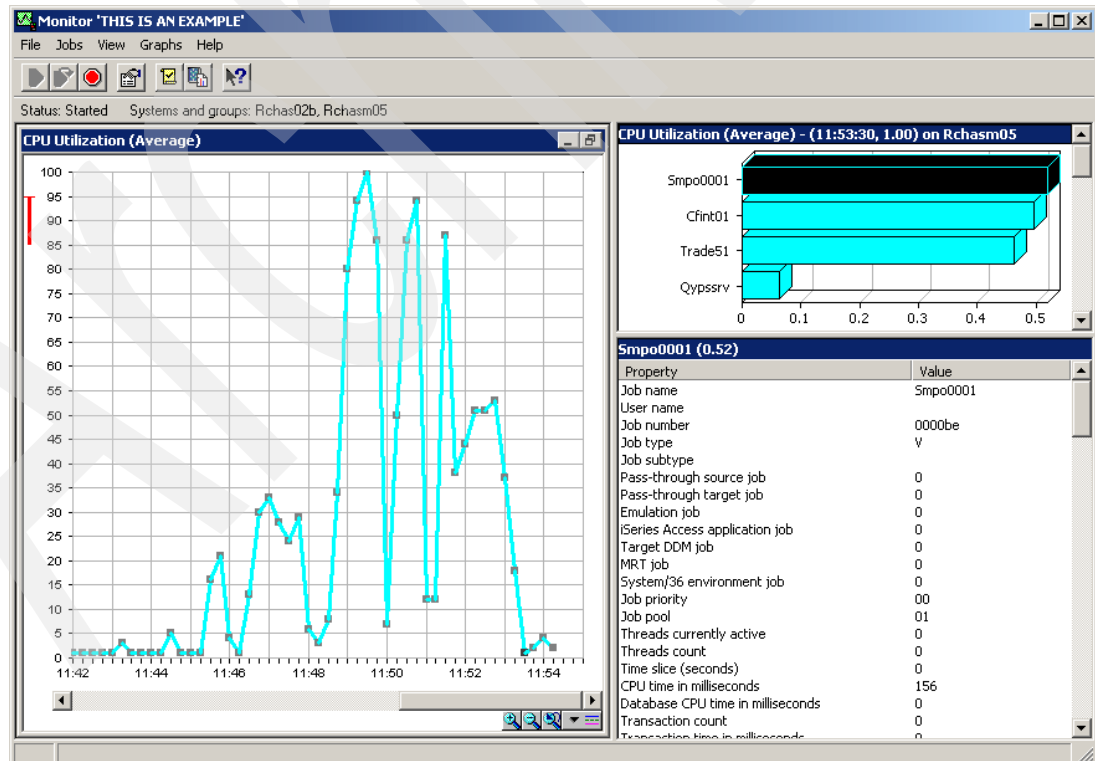


Figure 5-26 Using the iSeries Navigator for performance problem

This graph indicated that CPU utilization was bouncing up and down. The major task that was using the CPU was SMP0001, which handles the moving of data between the main storage and disk units. When we want to use the CPU for processing the data instead of moving it to and from disk, we have to reduce the rate of faulting.

There are two options to reduce the amount of faulting in a memory pool:

- ▶ Reduce the activity level
- ▶ Add more main storage

In this case, the only usable option was to increase the pool size. If we reduced the activity level, we would increase the amount of faulting and waste even more CPU.

After increasing the pool size to 1024 MB, the workload completed in 24 minutes and 35 seconds, but the users were still not happy. Figure 5-27 shows the different system behavior. There were no thread state transitions to the ineligible state, but the faulting rate was reduced to half of its previous value that is shown in Figure 5-24 on page 190.

| Work with System Status | | | | | | | | ASM05 | | |
|------------------------------|----------|-------------------------------|-------|------------|-------|-------------|-------|----------|----------|--------|
| | | | | | | | | 07/20/04 | 12:03:17 | |
| % CPU used : | 30.5 | System ASP : | | | | | | 157.9 | G | |
| % DB capability : | 4.5 | % system ASP used : | | | | | | 29.9193 | | |
| Elapsed time : | 00:08:06 | Total aux stg : | | | | | | 157.9 | G | |
| Jobs in system : | 2459 | Current unprotect used . . : | | | | | | 4720 | M | |
| % perm addresses : | .007 | Maximum unprotect : | | | | | | 4720 | 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 | 535.66 | 227.24 | +++++ | .0 | .0 | .0 | .3 | 8.5 | .0 | .0 |
| 2 | 6983.48 | .45 | 151 | .0 | .0 | .2 | 2.1 | 310.0 | .0 | .0 |
| 4 | 1528.75 | .01 | 10 | .0 | .0 | .0 | .0 | .3 | .0 | .0 |
| 5 | .25 | .00 | 1 | .0 | .0 | .0 | .0 | .0 | .0 | .0 |
| 6 | 1024.00 | .31 | 130 | .5 | 1.4 | 238.3 | 1707 | 2580 | .0 | .0 |
| | | | | | | | | | | |
| | | | | | | | | | | Bottom |
| ===> | | | | | | | | | | |
| F21=Select assistance level | | | | | | | | | | |

Disk utilization percentages were still relatively high but they were changing to a better state as shown when comparing the data in Figure 5-25 on page 192 to Figure 5-28.

| Work with Disk Status | | | | | | | | | | ASM05 |
|---|------|----------|--------|---------|------------------|----------|-----------|----------|-----------|-------------------|
| | | | | | | | | | | 07/20/04 12:03:37 |
| Elapsed time: 00:10:26 | | | | | | | | | | |
| Unit | Type | Size (M) | % Used | I/O Rqs | Request Size (K) | Read Rqs | Write Rqs | Read (K) | Write (K) | % Busy |
| 1 | 6718 | 13161 | 28.4 | 42.6 | 21.1 | 18.5 | 24.0 | 27.4 | 16.3 | 48 |
| 2 | 6718 | 13161 | 28.4 | 45.5 | 21.1 | 20.2 | 25.3 | 27.9 | 15.6 | 48 |
| 3 | 6718 | 17548 | 28.3 | 51.3 | 22.7 | 21.3 | 29.9 | 32.2 | 16.0 | 40 |
| 4 | 6718 | 13161 | 28.4 | 46.3 | 20.3 | 19.2 | 27.0 | 27.8 | 14.9 | 47 |
| 5 | 6718 | 13161 | 28.3 | 45.6 | 22.0 | 19.5 | 26.1 | 30.4 | 15.7 | 50 |
| 6 | 6718 | 17548 | 28.3 | 49.4 | 24.2 | 21.6 | 27.8 | 33.1 | 17.4 | 44 |
| 7 | 6718 | 13161 | 33.3 | 39.8 | 20.6 | 18.1 | 21.6 | 25.5 | 16.5 | 32 |
| 8 | 6718 | 13161 | 33.2 | 40.9 | 20.2 | 17.8 | 23.0 | 25.9 | 15.7 | 32 |
| 9 | 6718 | 13161 | 33.2 | 43.6 | 19.2 | 18.2 | 25.4 | 25.4 | 14.7 | 30 |
| 10 | 6718 | 13161 | 33.0 | 41.2 | 20.0 | 18.1 | 23.1 | 25.6 | 15.7 | 30 |
| 11 | 6718 | 17548 | 28.3 | 49.7 | 24.6 | 22.0 | 27.7 | 34.1 | 17.0 | 29 |
| | | | | | | | | | | Bottom |
| Command | | | | | | | | | | |
| ====> | | | | | | | | | | |
| F3=Exit F5=Refresh F12=Cancel F24=More keys | | | | | | | | | | |

Figure 5-28 WRKDSKSTS with the pool size of 1024 MB

Separate different workloads to separate memory pools

We decided to separate the two different workloads from each other by directing the jobs to separate memory pools. The workloads were the server jobs named QSQSRVR that ran in subsystem QSYSWRK and the WebSphere Application Server TRADE51 that ran in subsystem TRADE51. Since the TRADE51 was already running in a pool of its own, we decided to point the server jobs to a pool of their own.

We carved out an amount of memory for the server jobs pool by using the WRKSHRPOOL command. Then we used the Display Active Prestart Jobs (DSPACTPJ) command as shown in Figure 5-29 to determine the amount of activity levels needed.

```

                                Display Active Prestart Jobs                                ASM05
                                                                                   07/20/04 12:03:37
Subsystem . . . . . : QSYSWRK      Reset date . . . . . : 07/20/04
Program . . . . . :  QSQSRVR      Reset time . . . . . : 13:27:26
Library . . . . . :   QSYS        Elapsed time . . . . . : 0000:16:12

Prestart jobs:
Current number . . . . . : 90
Average number . . . . . : 36.1
Peak number . . . . . : 95

Prestart jobs in use:
Current number . . . . . : 72
Average number . . . . . : 33.2
Peak number . . . . . : 95

More...

Press Enter to continue.

```

Figure 5-29 Using DSPACTPJ command for setting the pools activity level

We directed the QSQSRVR jobs to a pool of 500 MB and an activity level of 95 and ran the workload again. This time the CPU was running at 99% utilization, but there was no paging nor faulting on memory pools three and six. Figure 5-30 shows the system main storage usage.

```

                                Work with System Status                                ASM05
                                                                                   07/20/04 12:35:42
% CPU used . . . . . : 99.9      System ASP . . . . . : 157.9 G
% DB capability . . . . : 16.7    % system ASP used . . . : 28.5238
Elapsed time . . . . . : 00:06:27 Total aux stg . . . . . : 157.9 G
Jobs in system . . . . . : 2473   Current unprotect used . : 2575 M
% perm addresses . . . . : .007   Maximum unprotect . . . : 2616 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   535.66   226.86  +++++  .0   .0    .0   .0    6.6   .0   .0
  2   6483.48    .31   151    .0   .0    .1   .2   52.9   .0   .0
  3    500.00    .00   95    1.2  3.6    .5   .6  267.1   .0   .0
  4   1528.75    .00   10    .0   .0    1.2  1.4  24.2   .0   .0
  5     .25    .00    1    .0   .0    .0   .0    .0   .0   .0
  6   1024.00    .31  130    .0   .0    1.6  2.6  5108   .0   .0

Bottom

====>
F21=Select assistance level

```

Figure 5-30 WRKSYSSTS display after separating the QSQSRVR jobs to a pool of their own

Since the data resided in the main storage, the disks were not utilized over the guideline as shown in Figure 5-31.

| Work with Disk Status | | | | | | | | | | ASM05 |
|---|------|----------|--------|---------|------------------|----------|-----------|----------|-----------|-------------------|
| Elapsed time: 00:06:27 | | | | | | | | | | 07/20/04 12:35:42 |
| Unit | Type | Size (M) | % Used | I/O Rqs | Request Size (K) | Read Rqs | Write Rqs | Read (K) | Write (K) | % Busy |
| 1 | 6718 | 13161 | 26.8 | 34.7 | 4.0 | .3 | 34.4 | 5.3 | 4.0 | 3 |
| 2 | 6718 | 13161 | 26.8 | 44.8 | 4.0 | .3 | 44.4 | 4.0 | 4.0 | 2 |
| 3 | 6718 | 17548 | 26.8 | 34.5 | 4.0 | .1 | 34.3 | 7.6 | 4.0 | 2 |
| 4 | 6718 | 13161 | 26.8 | 27.5 | 4.1 | .1 | 27.3 | 7.2 | 4.1 | 1 |
| 5 | 6718 | 13161 | 26.8 | 19.6 | 4.2 | .4 | 19.2 | 4.4 | 4.2 | 2 |
| 6 | 6718 | 17548 | 26.8 | 43.4 | 4.0 | .3 | 43.1 | 8.9 | 4.0 | 2 |
| 7 | 6718 | 13161 | 31.9 | 46.7 | 4.0 | .1 | 46.6 | 4.0 | 4.0 | 2 |
| 8 | 6718 | 13161 | 31.8 | 46.7 | 4.0 | .1 | 46.5 | 4.7 | 4.0 | 3 |
| 9 | 6718 | 13161 | 31.9 | 65.7 | 4.0 | .1 | 65.6 | 4.6 | 4.0 | 4 |
| 10 | 6718 | 13161 | 31.6 | 46.9 | 4.0 | .1 | 46.8 | 7.2 | 4.0 | 4 |
| 11 | 6718 | 17548 | 26.8 | 44.6 | 4.0 | .1 | 44.4 | 5.7 | 4.0 | 2 |
| | | | | | | | | | | Bottom |
| Command | | | | | | | | | | |
| ====> | | | | | | | | | | |
| F3=Exit F5=Refresh F12=Cancel F24=More keys | | | | | | | | | | |

Figure 5-31 WRKDSKSTS command after separating the QSQRVR jobs to a pool of their own

The graphical system monitor displayed in Figure 5-32 looks completely different from the one shown in Figure 5-26 on page 192, where the system was suffering from the heavy faulting in the memory. The workload completed in 9 minutes and 30 seconds.

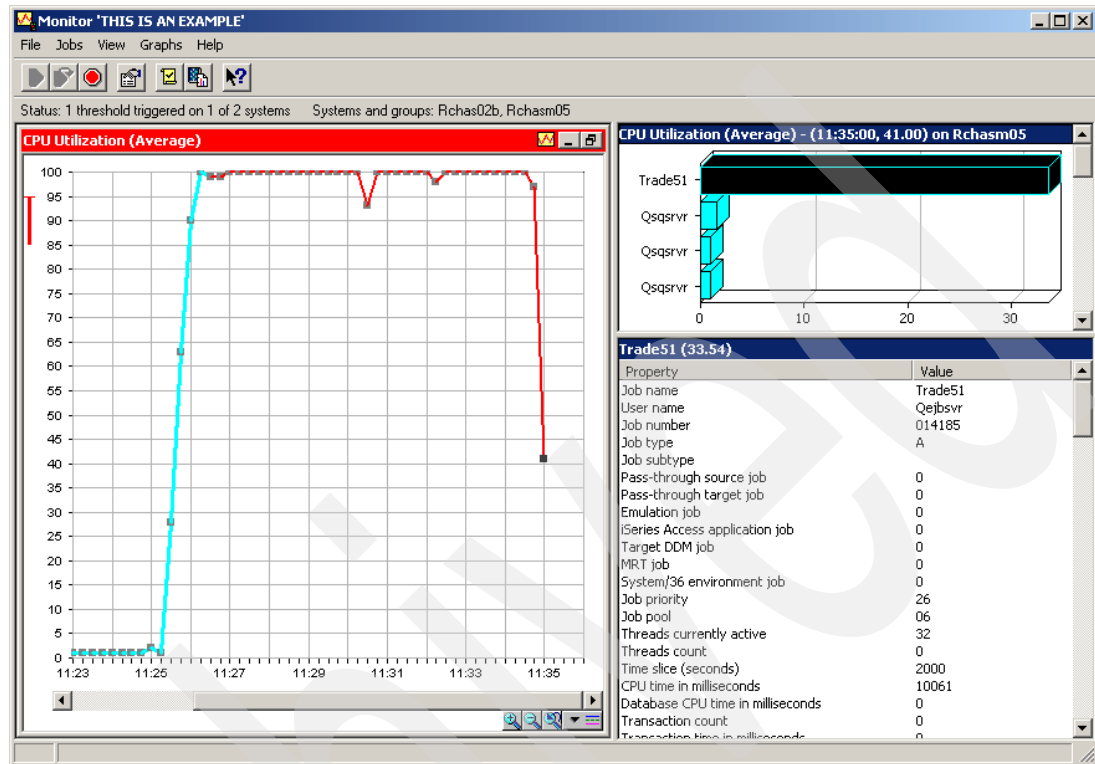


Figure 5-32 iSeries Navigator view of the workload on a tuned system

Archived

Tuning the IBM HTTP Server (powered by Apache)

Through the various transitions of Web servers for the iSeries, the fundamental base in which to work from has, for the most part, always been the IBM HTTP Server. Underneath the covers, it has more or less been equated to the famous and now universally accepted Apache Web server.

The Apache Web server has been around for nearly 10 years and is currently the most widely used among the top servers. Furthermore, it is freeware, open-source software, and highly configurable and easily extendable. You can find additional information about Apache on the Web at:

<http://www.apache.org>

So why do we call it IBM HTTP Server (powered by Apache)? This is because the Apache on the iSeries is based on the open-source server code (version 2.0), but not the actual source code. Additional enhancements for the iSeries have been made. This chapter focuses on how to use the HTTP Server (powered by Apache), and not solely the previous versions of the IBM HTTP Server. Reference to the HTTP Server in this chapter imply powered by Apache.

This chapter goes deep into the performance issues centered around the HTTP Server. In particular, it sifts through the following areas:

- ▶ Web server performance
- ▶ Caching
- ▶ Persistent connections
- ▶ Secure Sockets Layer (SSL) and Transport Layer Security (TLS)
- ▶ HTTP server tuning options
- ▶ Static content location
- ▶ References

6.1 Web server performance

Anytime you surf the Web, you access several Web sites, each of which contains one to many components. The great cloud of the Internet can definitely be hazy and obscure, just as long as you can still order your DVDs or buy a new shoes via the Web. In any case, the components of a Web serving environment can be boiled down into three parts: a client, a network, and a server. It can be way more complicated than that, but in essence these are the parts that are involved.

And where does the HTTP server fit into this? The HTTP server fits on the back side as the server, unless you consider the HTTP server as a client, requesting information from another server, which is also possible. For the sake of simplicity, think in terms of client, network, and server. With this simple mind-set, the performance problems that may arise can be attributed to one or all of these areas. See Chapter 1, “Performance concepts and factors” on page 1, for more discussion about this topic.

6.1.1 The server

When you enter into the server arena, you face a challenge in attempting to come up with a nice and simple plan to gain performance boosts. Why is this? Because the variables, flavors, and complexities of the server are as varied as the patterns of a snowflake. That's the point about the server. The idea is to handle all sorts of applications, both large and small, dynamic and static.

Still, you can narrow down your search for performance increases by remaining steadfast to the common factors of application performance; resources, database, and the application itself. In a more robust and complex environment, you have split this application workload into several components: a Web server, an application server, and a database server. This combination of servers can increase or decrease depending on the availability of the application and the volume of transactions for it.

How does the HTTP server fit into this mix? Simple, it is the Web server portion. And the application server is WebSphere. And for the iSeries, you can either have the database server running locally or remotely, depending on the topology of your companies' data layer.

As you may have guessed, the server layer is, in almost all cases, the layer in which you spend the majority of your time debugging performance issues. It has the most volatile components, the pieces and parts that you have created, tweaked, or plugged in, to provide an effective and reliable application or Web site.

6.2 Caching

Caching is one area of interest when it comes to increasing performance. In particular, caching allows resources and pages to be available in memory, as opposed to having to be retrieved from disk. In most cases, this can save a considerable amount of response time for the user.

Caching on the iSeries is setup using a hash table. Each object (GIF, HTML, XML, JavaScript, or any file required for a page) in the HTTP request is indexed within the hash table. For each request, a lookup is done on the hash table to determine if you set the object to be cached (*static caching*). If so, the object is retrieved from memory via the hash table lookup and returned to the user. If the object has not been already used, then further processing is done to retrieve the object from its original location. After this is done, the object is then cached for future use (*dynamic caching*).

6.2.1 Dynamic and local (static) caching

In terms of caching, you can define specifically which objects are cached (static or local caching), or allow the system to determine which objects to cache. In dynamic caching, the system determines which files are most commonly used.

And which one is best: local or dynamic caching? It depends on the nature of your application or Web site. On smaller Web sites, dynamic caching works well because the cache size and number of objects in cache are relatively small.

When you move into a larger Web site, you have to be more careful with dynamic caching. The system can spend more time managing the cache than serving the pages. This is where the balance must be made. Therefore, on some larger sites, static caching may be better, especially if you have several heavily used objects. If you choose dynamic caching, keep an eye on the page faults (Work with System Activity (WRKSYSACT) command) in case the cache is consuming too much memory.

Cached pages (or objects for the page) take less time to serve since they are in memory. Non-cached pages take up more time because such additional resources as input/output (I/O) disk and central processing unit (CPU) are required. For example, a simple read from disk may take several milliseconds, but a read from memory is less than or equal to one millisecond. Furthermore, additional CPU cycles are involved with a disk read as opposed to a memory read.

Why not load everything into memory? Some day that may be the case. But for now, memory is more expensive than disk so we must balance the need between the two. The trick to effective caching is to determine which objects and pages will be most frequently used. Caching objects that are used only once or twice is an ineffective use of memory.

A simple rule of thumb is to load all static pages (plain HTML pages that have no dynamic information) and graphics into cache. However if the pages or graphics are rarely used, there is no benefit to doing this. In the event that you attempt to cache dynamic content (HTML pages rendered from JavaServer Pages (JSP) code such as a stock market trade application), the content is never truly cached for the next time it is changed (such as the stock market trade value goes up .20). The page must be removed from cache for the content is different. Pages with dynamic content are not a good choice for caching. However, some of the objects within the dynamic page may be suitable for caching.

6.2.2 Proxy caching

Many organizations use a proxy mechanism when sending and receiving information between the Internet and intranet. A *proxy service* masks or hides the actual IP addresses within your internal network from the outside world, preventing malicious attacks to your network. So when you connect to the Internet from your internal network, the IP address the Internet sees is not the real IP address of your PC on the internal network.

With proxy caching, the Internet Web pages requested from within your internal network (intranet) are cached for future use. Frequently used documents from the Internet can be cached, improving performance. In addition to this, a proxy service can be used to log client requests. This will allow you to capture client information such as IP address, day/time and Uniform Resource Locator (URL) request. This information can be used as metrics for reporting purposes on your application or Web site.

Another form of proxy is *reverse proxy*. This type of proxy is used to prevent the outside Internet from directly accessing your internal network, or parts of it. If caching is turned on for

reverse proxy, it can reduce traffic on your network by having cached files and documents returned instead of sending the request all the way back to its original location.

6.2.3 Tuning cache

The cache file methods are usually configured through the administration graphical user interface (GUI). To configure, for example, all the GIF files in your document root to be cached at server startup, follow the steps as explained here and shown in Figure 6-1.

1. Start IBM Web Administration for iSeries (the HTTP Admin GUI) and click **IBM HTTP Server for iSeries**.
2. Select the **Manage** tab at the top of the page. Make sure the **HTTP Servers** subtab is selected.
3. From the Server list, select the server that you want to manage. From the Server area list, select **Global configuration**.
4. In the left panel under Server Properties, select **System Resources**.
5. Click the **Caching** tab.

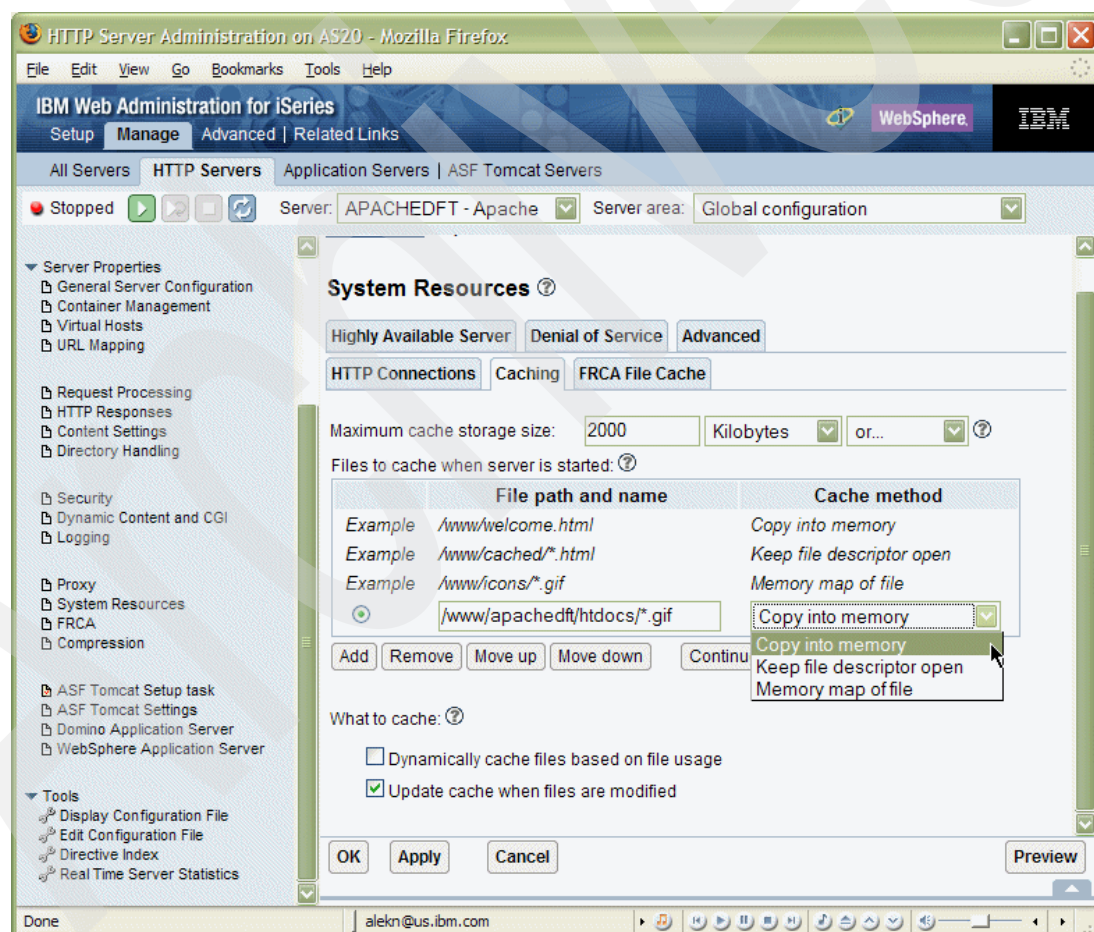


Figure 6-1 Setting the caching option

On the Caching tab, follow these steps:

- a. In the Maximum cache storage size field, type the memory size you want to configure for the files to cache. Remember that this size is the main storage size used by all the files that you decide to preload into memory. For this example, we use the default 2000.

- b. Under the Files to cache when server is started table, click **Add**.
- c. In the File path and name column, enter the file path for the static data the Web server will cache at server startup. For this example, we cache the images (*.gif) files of the document root /www/apachedft/htdocs/*.gif.
- d. In the Cache method column, select the mechanism used to cache the files. For this example, we select **Copy into memory**.
- e. Click **OK**.

6.2.4 Fast Response Cache Accelerator

Fast Response Cache Accelerator (FRCA) refers to providing a cache underneath the machine interface (MI) layer for static and dynamic files. It provides a shortcut for the retrieval of files from their original location, regardless of files in a normal caching mechanism above the MI layer.

FRCA then provides a performance improvement by reducing the amount CPU cycles needing to handle the request. Why is this important? It can serve files up to seven times faster than directly from the integrated file system (IFS). And it is four times faster than local caching on the HTTP Server. For details on the performance characteristics of FRCA, see *iSeries Performance Capabilities Reference Version 5, Release 2*, SC41-0607. This document provides release-to-release performance information about the iSeries server.

Originally FRCA was named Adaptive Fast Path Architecture (AFPA). It provided an architecture to improve both performance and capacity of Web page serving. The HTTP Server (powered by Apache) has bought into this technology. In doing so, it gives users of the iSeries a dramatic increase in performance.

How is this done? The key to implementing FRCA is in the directives it uses, which are located in the HTTP configuration file. You can set up FRCA to listen on each port, and either cache specific static files or a group of files using the wildcard asterisk (local caching), or cache dynamic content through a URI (reverse proxy caching). The key is that FRCA is setup to be used both as a local cache and a reverse proxy cache.

For more information about FRCA, see *HTTP Server (powered by Apache): An Integrated Solution for IBM @server iSeries Servers*, SG24-6716.

6.2.5 Network File Cache

With V5R2, Network File Cache (NFC) gives you the ability to store and retrieve files in a more efficient way. NFC provides a way to incorporate a partitioned file system specifically for System Licensed Internal Code (SLIC) tasks. The file system will handle all MI SLIC tasks such as open, read, write, and close.

The NFC file system size is setup by default at 10 MB and runs out of the base user pool. NFC is used in conjunction with FRCA. To change the size of NFC, you use the Change TCP/IP Attributes (CHGTCPA) command. Change the NFC parameter to something such as NFC(*YES 300 10). This setting is determined by the total size of the Local FRCA Caches for all HTTP Server instances. For two HTTP Server instances that have their Local FRCA Cache each set to 2 MB, the NFC file system size is 4 MB.

For more information of NFC and FRCA, see *HTTP Server (powered by Apache): An Integrated Solution for IBM @server iSeries Servers*, SG24-6716.

6.3 Persistent connections

Every time you enter a URL on the address line of your browser and press Enter, or click a link from a Web page, you open a connection to an HTTP server. You make a connection between your local client browser and the remote HTTP server somewhere on the Internet. Furthermore, for each file needed on the Web page, a separate connection is required.

As you can imagine, many connections are opened and closed just for one user alone as they traverse a Web site. When additional hundreds of users are applied, the situation worsens. This is where persistent connections come into play. Ideally, performance increases can be found by allocating a dedicated thread for each client for a duration of a client's session.

However, there are some pitfalls with using only the persistent connections as discussed in the following sections.

6.4 Secure Sockets Layer and Transport Layer Security

SSL has become the industry standard for communicating securing over the Internet. TLS is the latest update to SSL and provides even greater detail to security requirements.

In terms of performance issues for SSL and TLS, if the Web site or Web pages is setup to use client authentication, the server requests have an increased overhead due to the additional SSL client certificate. This in turn reduces performance for every page using SSL and TLS.

6.5 HTTP server tuning options

You can choose from many options when it comes to altering the performance characteristics of your HTTP Server. Finding the right ones to tweak can sometimes be more of an art than a science.

Here is a list of some areas to focus on when tuning your HTTP server:

- ▶ Number of threads to process the requests
- ▶ Threads and asynchronous I/O
- ▶ Triggered Cache Manager
- ▶ mod_deflate tool
- ▶ Logging
- ▶ HostNameLookups
- ▶ KeepAliveTimeout
- ▶ TCP buffer size
- ▶ Denial of service

6.5.1 Threads and asynchronous I/O

The HTTP Server (powered by Apache) has its own multi-process model. Each HTTP server starts two (or three) processes under the QHTTPSVR subsystem:

- ▶ The manager process
- ▶ The primary process
- ▶ The backup process, when configured with the HotBackup directive

Each child process maintains its own thread pool independently.

This setting is one of the most important attributes of the HTTP Server (powered by Apache). It allows you to specify how many threads each child process is allowed to use. The default value is the same as the value for the maximum number of threads found on the Global Server Settings form. The directive is *ThreadsPerChild*.

That is, you can set this parameter at the Global Server Setting to be the default value at startup for all your Web servers (original and powered by Apache) as shown in Figure 6-2. Then you have the option to override this Global Server Setting for each HTTP Server (powered by Apache).

You can *only* configure the maximum number of threads that the server opens at startup. The HTTP Server (powered by Apache) always starts with the maximum number of configured threads. The HTTP Server (powered by Apache) implementation does not use the minimum thread value.

When no threads are available, the Web server response time, from the client point of view, is impacted since the request takes longer because of the lack of available threads. Setting this number too low impacts server performance since the client request cannot be processed until the Web server finds an available thread. However, setting the maximum number of threads too high, in general, requires more system resources to keep those threads available for use. There is no optimum value for this setting.

With the HTTP Server (powered by Apache) implementation, the HTTP Server processes communications requests asynchronously. In this asynchronous I/O model, threads are only involved in processing when work is to be done. Threads are dispatched to perform work as required. When not performing work, the threads are returned to a pool of available threads. This makes the server process more efficient and improves performance by using the thread resources better. Asynchronous I/O also makes the server more scalable to support a high number of users especially when combined with persistent connections. We recommend that you keep the default value of *on* (or enabled). The directive is *AsyncIO*.

Tip: Asynchronous I/O is one of many enhancements to the standard Apache server as delivered to IBM Rochester by the Apache Software Foundation (ASF). This is one of the many reasons that the parenthetical phrase (powered by Apache) means integration.

Setting the maximum number of threads for all HTTP servers

you configure the minimum and maximum number of threads on the Global Server Settings display (Figure 6-2) for all your HTTP servers. To set the maximum number of threads, follow these steps:

1. Select the **Advanced** tab as shown in Figure 6-2.
2. In the left panel, under Global Settings, select **Global Server Settings**.
3. In the right panel, for Number of threads, in the Maximum field, type the maximum number of threads.

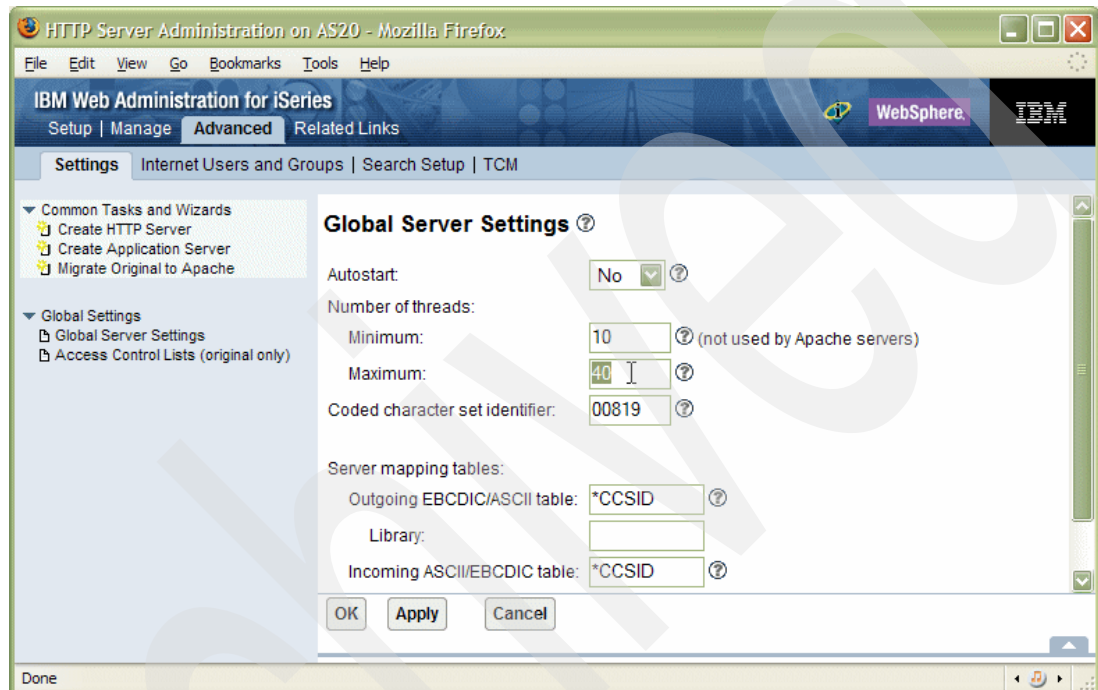


Figure 6-2 Setting the number of threads globally

Setting the maximum number of threads for a single HTTP server

You can also set the maximum number of threads and the option to use asynchronous I/O for each individual HTTP Server (powered by Apache) as shown in Figure 6-3. Follow these steps:

1. Select the **Manage** tab.
2. Select the server you want to manage. For Server area, select **Global configuration**.
3. In the left panel, under Server Properties, select **System Resources**.
4. Click the **Advanced** tab.
5. In the right pane, scroll down and change Number of threads to process requests to override the Global Server Settings. Click **OK**.

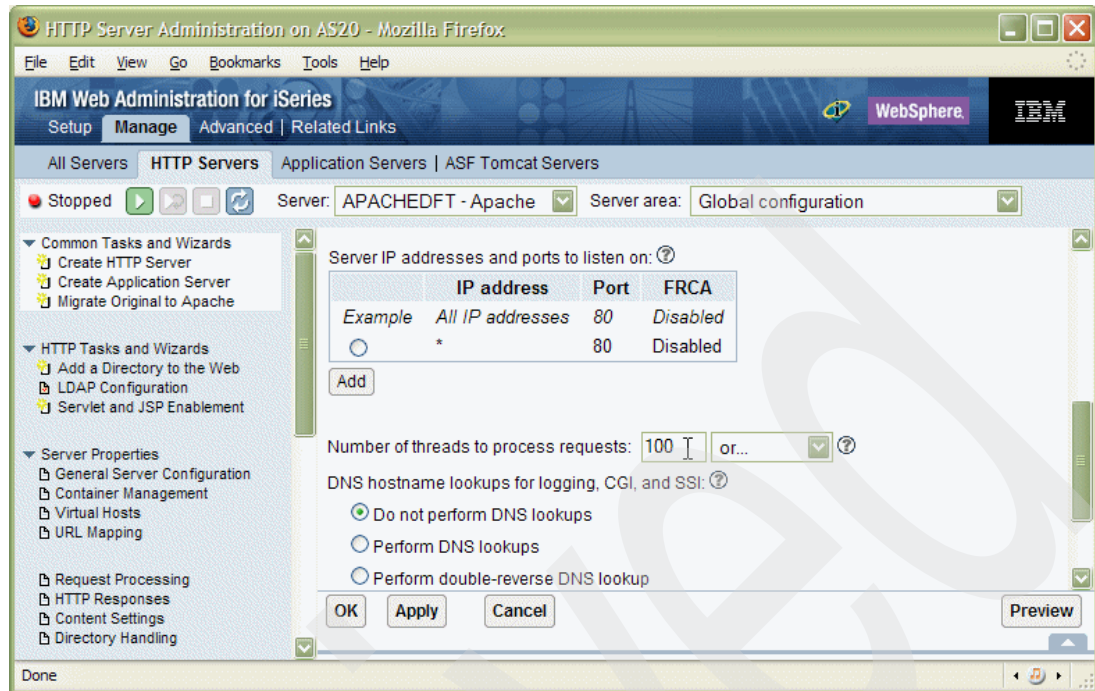


Figure 6-3 Setting the number of threads for a specific server

6.5.2 Triggered Cache Manager

Each time the client requests a page, the HTTP request must go through this process:

1. It goes through the network and passes each communication component such as firewall, routers, proxies, and so on.
2. It goes through the HTTP Server (powered by Apache) to identify that the request must be redirected to an application server.
3. The application server must in turn call the application.
4. The application processing the request may have to rely on many large object (LOB) database queries (or any other required tasks) to generate the dynamic content of the HTML page.
5. Even after the HTML is generated, this response must flow all the way back down the line.

If a different client or even the same client requests the information again, the whole process must be repeated. Even if the application has an application cache that can reduce the number of queries into the LOB database, the amount of processing required to place the object stored in the application cache can be extremely large.

If you extend this environment to one with hundreds or even thousands of clients, CPU cycles (alone) required to process the requests increase considerably. At this point, it may be better if the implementation included a mechanism to serve client requests without going through the whole process every time.

In this implementation, you create the dynamic content in a proactive fashion and copy it to the iSeries IFS or perhaps a router with a cache mechanism. Either way, now when the client sends the request, dynamic content is already cached in the iSeries IFS or router. In this way, the request can be processed with less demand on server resources and the response time for the client improved.

Triggered Cache Manager (TCM) is a component in the iSeries server that was created exactly for this purpose. This component is packaged in the IBM HTTP Server for iSeries, 5722-DG1 Option 1. TCM is a Transmission Control Protocol (TCP) server. This server may be used in conjunction with Web servers and Web document caching agents to keep Web sites running at peak performance.

TCM is:

- ▶ A cache manager, not a cache or cache server
- ▶ Based on trigger messages

This means that you must set up application triggers for the server to work. These messages are sent to the TCM via the HTTP protocol.

- ▶ A stand-alone server that can work with multiple types of caching mechanisms, for example caching routers, proxy caches, and so on

It is useful in the environment that we describe here for the HTTP Server (powered by Apache), but it can be used for the HTTP Server (original), Lotus® Domino®, or WebSphere Application Server environments as well.

TCM is proactive (based upon updates to an LOB database) in the update of Web content. TCM causes the Web content to be dynamically regenerated and then placed in a location (usually the iSeries IFS) that was configured to be served as static content by your Web server. The TCM is most effective for a Web site that has a large number of requests for content that is somewhat constant, but changes frequently.

One of IBM's first uses of the TCM concept was to drive the 1996 Summer Olympic Games Web site. For example, a downhill skiing results page (mostly HTML) was composed of hundreds of dynamic items including names, times, scores, and so on for a particular event. For every request of that page, consider if the application server had to re-run all the database queries to dynamically calculate the content. That level of activity could have weighed heavily on the server with many repetitive actions. Only when the application LOB data was updated with new results, then the application server was used to update a standard results page to be served from a "static" portion of your Web site. This tremendously reduced the burden on the application server.

For more information, read *HTTP Server (powered by Apache): An Integrated Solution for IBM @server iSeries Servers*, SG24-6716.

6.5.3 mod_deflate

mod_deflate is a tool used to compress files server from the HTTP Server. This is done through the configuration files and is the equivalent of Apache's mod_gzip. When Web content or files are compressed, it reduces the amount of bandwidth required to send the information. The data is eventually decompressed on the client side in the Web browser. Keep in mind that additional resources are consumed to compress and decompress the content. Unless you expect or suspect bandwidth issues, mod_deflate is not necessary.

6.5.4 Logging

There are several different areas of logging to consider for the HTTP Server. And depending on your needs or performance problem, you may or may not want them collecting detailed information. Remember, the more you collect, the more resources are used to write to disk.

Access logs

One of the most important tuning options when it comes to logging has to do with the Access Log. By default, this log is turned on, and all HTTP requests are captured.

To turn off the HTTP server Access Log, follow these steps:

1. Access your HTTP server configuration page in IBM Web Administration for iSeries.
2. Under Tools, click **Edit Configuration File**.
3. Search for a line with the text CustomLog (see Figure 6-4). Comment out this line by placing the pound symbol (#) in front of the line.

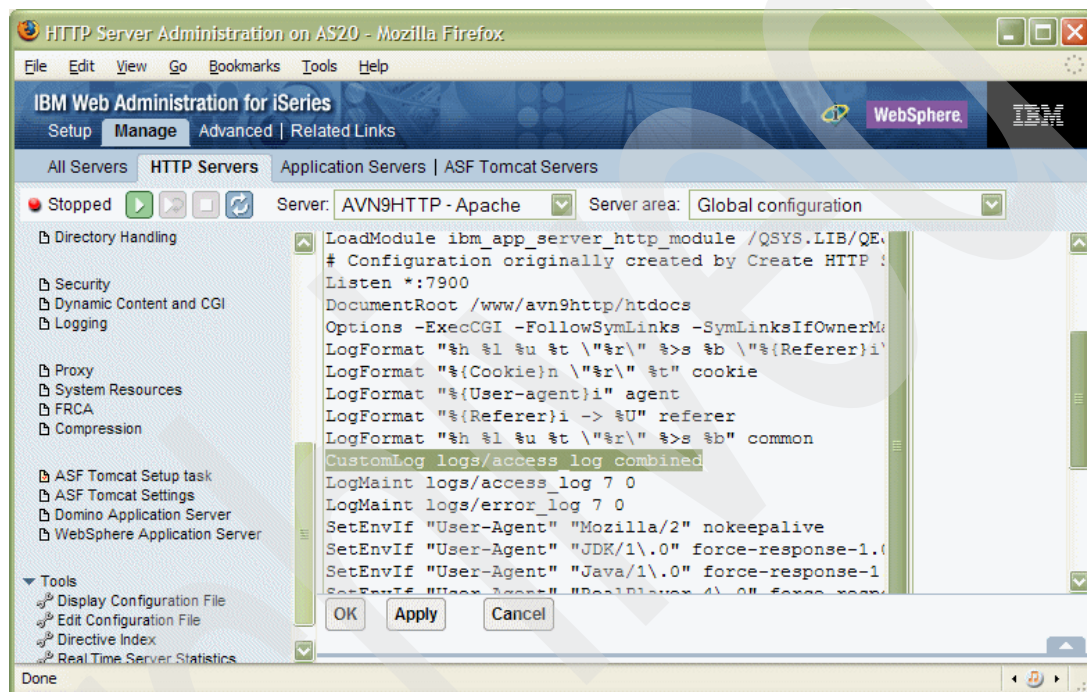


Figure 6-4 Disabling the access log

4. Save and close the configuration file.
5. Restart the IBM HTTP Server.

The default value for the access logs is *enabled* for an incoming HTTP request. The recommended value for performance reasons is to disable the access logs.

Error logs

The error log for the HTTP Server can be setup at several different levels. Depending on that level, your performance can degrade. The bottom line is that the more you log, the more resources are consumed and the performance of your application decreases. Different levels of logging are:

- Emergency
- Alert
- Critical
- Error
- Warning
- Notice
- Informational
- Debug

And if your logging is set to *Debug*, then all messages are logged, from Emergency to Debug. This impacts your server the most when compared to setting it to Emergency.

Job logs

Another place to check for either abnormal errors or performance problems is in the HTTP server job log. The job description (JOBID) for the HTTP server is QZHBHTTP in the QHTTSPSVR library. By default, the user profile used is QTMHHTTP. You can change the message logging settings to allow more or less information to be recorded. Using the Change Job Description (CHGJOBID) command, you change the Text setting to *NOLIST, *MSG, or *SECLVL. *SECLVL is the most extreme and should be used with special attention.

6.5.5 HostNameLookups

HostNameLookups is a directive in HTTP configuration file that enables your HTTP server to perform reverse lookups, converting the IP address into a host name and domain and logging that information. With this directive on, you can track the usage of your Web site or determine possible network problems. The default is set to off, which saves on network traffic. However, high-volume Web sites should leave it to *off*. This prevents DNS lookups from occurring and saves time and resource.

6.5.6 KeepAliveTimeout

KeepAliveTimeout is another directive in the configuration file. It is used to determine how long a single TCP connection should remain open and available for a particular client if there is not activity from the client. This setting is used in conjunction with persistent connections.

What should you set KeepAliveTimeout to? If you set it too low, then the connections close too soon before another request can use it, causing additional resources to open a connection. If you set the value too high, then the server spends too much time holding onto open connections that are not being used. Thankfully, asynchronous I/O resolves most of the issues with this for as soon as a request is done with a connection it releases it, making it available for the next request. In the end, use the default value, unless for some reason persistent connections are not being used.

6.5.7 TCP/IP buffer sizes

The TCP/IP buffer size defines how much data can queue up, waiting for an application to receive (or send) it before any further data from the remote system is refused. The process of refusing further data is implemented at the TCP/IP protocol level and is called *flow control*.

Flow control can use significant CPU time, and results in additional network latency to wait for the signal to resume transmission. We recommend that you increase the buffer sizes from their default of 8 KB to avoid flow control under normal operating conditions. A larger buffer size reduces the potential for flow control to occur, improving efficiency of the CPU.

Larger buffer sizes also increase the amount of work that an erroneous client or a malicious attacker can queue up. You can use the buffer size as a simple means of limiting the resources that such conditions can consume. The idea is to avoid flow control, but not allow more work to buffer up than the system has the capacity to process.

The recommended buffer size to provide the best throughput depends upon several network environment factors. These include the types of switches and systems, acknowledgement timing, error rates, and network topology. When transferring large amounts of data, you may experience higher throughput by setting the buffer sizes up to 8 MB (which is the maximum).

This enables the data to keep flowing and allows the receiver to accept data in large segments.

Using the largest buffer size is not appropriate in most environments. If you only expect to serve requests and responses of modest size, then a larger buffer size will not improve your throughput. Also, the protocol for most socket servers is to process one request at a time for a given connection. Under such a protocol, the client (if properly following the protocol) won't queue up more than one request. Therefore, buffers that exceed the size of a single, large request are not required.

To review or revise the default TCP/IP buffer sizes, enter the CHGTCPA command on a command line and press F4 (Prompt). The buffer sizes are shown on the display as the TCP receive buffer size and the TCP send buffer size.

As a general guideline, we recommend that you set both the send and receive buffer sizes to 1 MB. To do this, enter the following command:

```
CHGTCPA TCPRCVBUF(1048576) TCPSNDBUF(1048576)
```

Note: These buffer sizes are used as the default for each TCP/IP conversation that is established. Applications may explicitly control the buffer sizes used for their specific connections by using the SO_SNDBUF and SO_RCVBUF socket options on the Set Socket Options, setsockopt(), API.

If you are using the IBM Toolbox for Java JDBC driver, you can configure the buffer sizes you want to use in the custom properties configured for the data source.

The connection between the HTTP plug-in and the application server is overridden to be 64K. Therefore, the buffer sizes specified in the CHGTCPA command are not affected.

6.5.8 Denial of service

Unfortunately, there are sometimes malicious attacks on your Web site. Hackers who try to break into your system are hoping to find that proverbial pot of informational gold. One such attack has to do with a *denial of service*, an attack that can create grand performance degradation of epic proportions. The attack intentionally slows down your Web site to hopefully break your firewalls and other security mechanism, allowing full access to your company's information.

One way to prevent this attack is to specify certain attributes to create a more secure environment. The increase in security causes a slight decrease in performance due to the tracking. Why? The settings that you make allow you to identify the possibility of an attack based on the frame size of your data. An attack may come when the frame size is different than the one it expects.

You must set up the following directives:

- ▶ **LimitRequestBody:** Maximum message body size. Limits the size of the HTTP request message.
- ▶ **LimitXMLRequestBody:** Maximum XML message body size. Limits the size of the XML message being requested. The default is 1 MB.
- ▶ **LimitRequestFields:** Maximum header fields. Limits the number of request header fields. The default is 100.
- ▶ **LimitRequestFieldSize:** Maximum header file size. Limits the size of the HTTP request header field. The default is 8190.

- **LimitRequestLine:** Maximum HTTP request line. Limits the size of the client's HTTP request line. The default is 8190.

You can configure the denial of service settings as explained here and shown in Figure 6-5:

1. Select the **Manage** tab.
2. For Server, select the server you want to manage. For Server area, select **Global configuration**.
3. In the left panel, under Server Properties, click **System Resources**.
4. Click the **Denial of Service** tab.
5. On the Denial of Service tab, enter the values for your environment. Click **OK**.

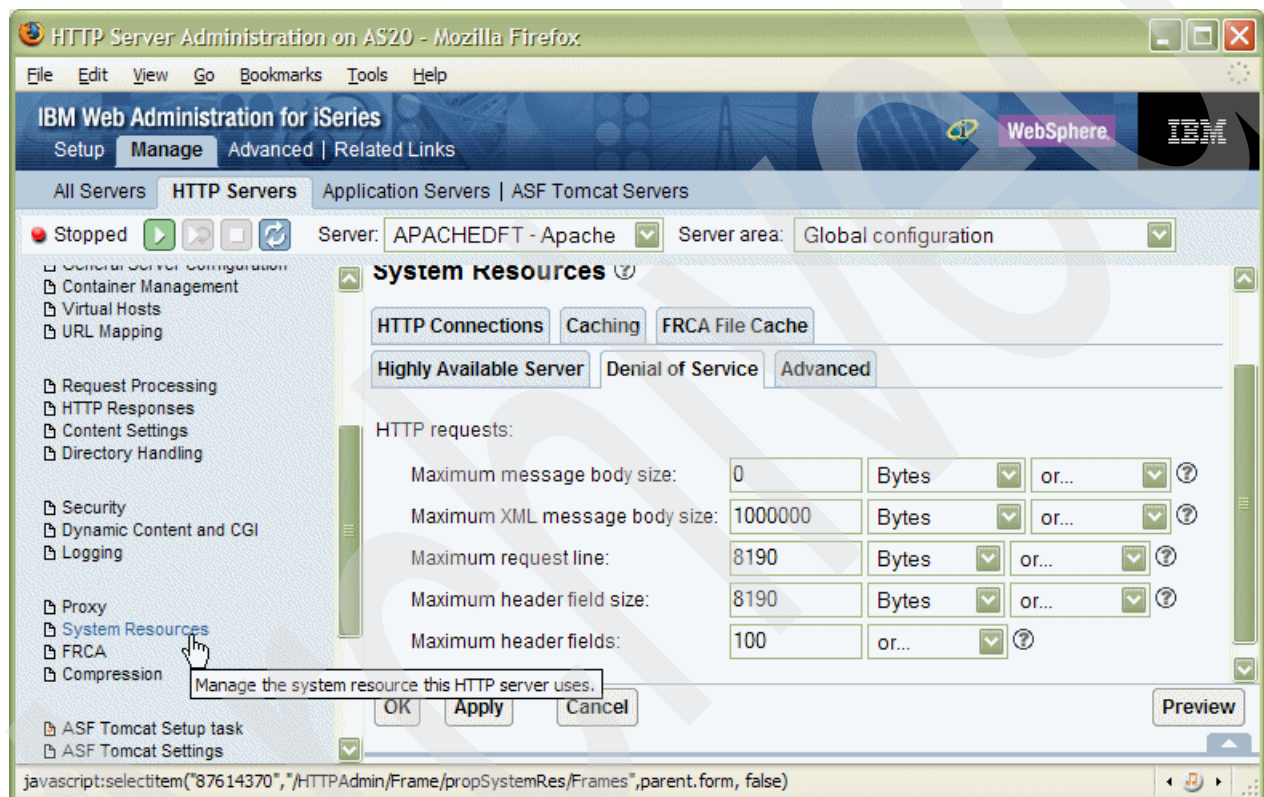


Figure 6-5 Setting the Denial of Service parameters

6.6 Static content location

Another important aspect to application and Web site performance in terms of the HTTP server is the location and storage of static content. Static content can consist of plain HTML files, GIF files, CSS files, or even PDF documents. For the most part, these files do not change or are not modified frequently. In some cases, they do not change for the entire life of your Web application.

This kind of data or these kinds of files should be positioned to be available to serve the user as quickly and efficiently as possible. It does not make sense to have them stored and retrieved from with WebSphere. Keep in mind, that WebSphere as an application server is geared toward serving up dynamic information, which changes frequently. Having to serve static information that almost never changes is a waste of resources and CPU.

Where is this static content served from? The best solution is to use the server that was made for this kind of information, a Web server such as the HTTP server (powered by Apache). This is fine, but how do you actually implement this type of solution? The key is to make sure the static content is stored in its own location or directory during the development stages. Then, when it comes time to deploy for system testing and production, the directory of static files can easily be deployed within the HTTP servers domain instead of WebSphere's.

Use the following steps to allow HTTP server to serve your static files instead of WebSphere:

1. Disable the WebSphere file-serving feature in the Web module:
 - a. Start the Java 2, Enterprise Edition (J2EE) development environment, for example WebSphere Development Studio Client for iSeries.
 - b. Import your application. It should be packaged as an Enterprise Archive (EAR) file.
 - c. In the J2EE perspective, expand **Web Modules**.
 - d. Double-click your **Web module**.
 - e. Web Deployment Descriptor opens. Click the **Extensions** tab at the bottom of the edit pane.
 - f. In the WebSphere Extensions pane, deselect the **File serving enabled** check box. Save your changes.

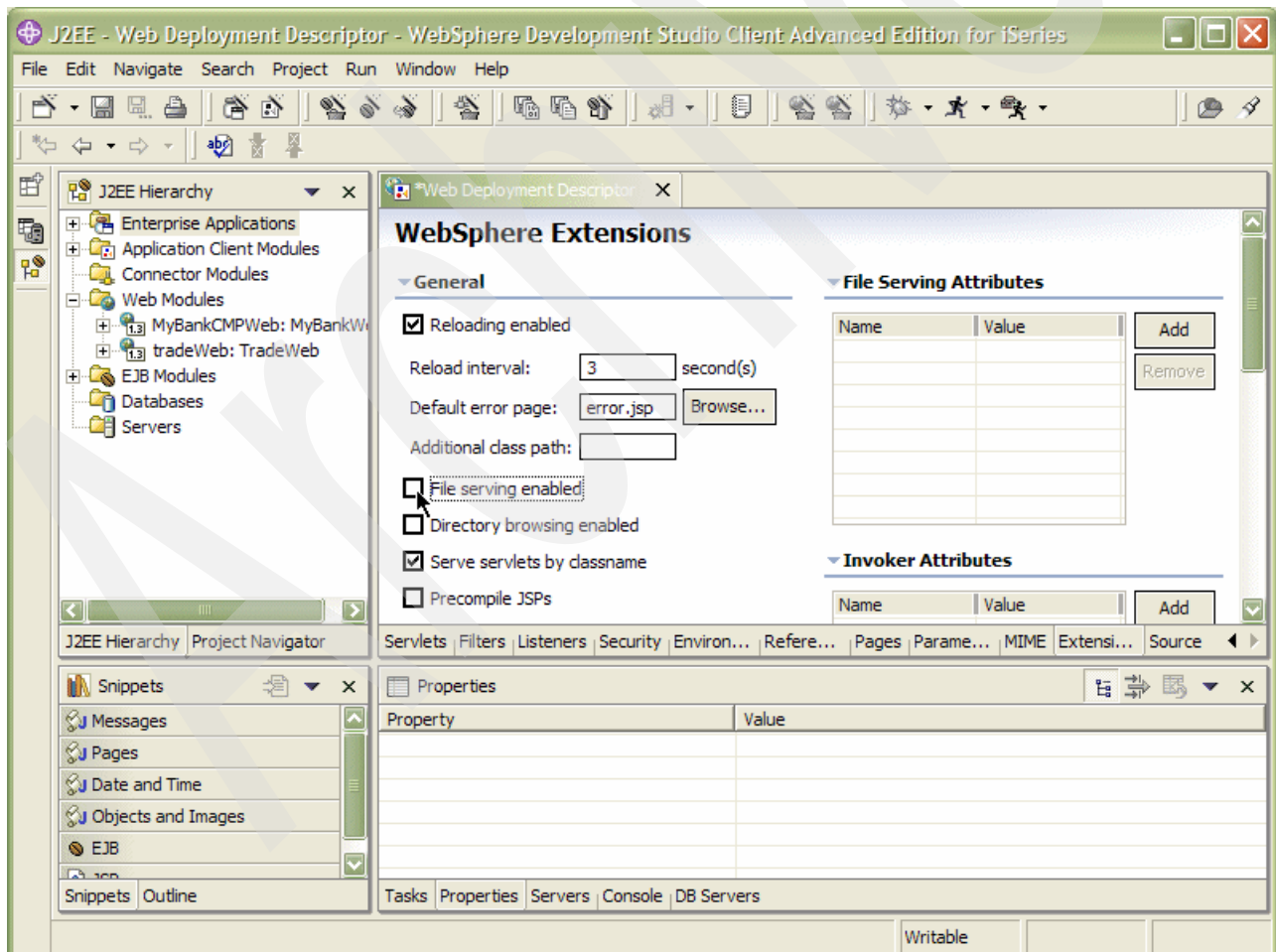


Figure 6-6 Disabling the file serving capability

- g. Export your EAR file from WebSphere Development Studio Client.

2. Install your EAR file on your application server machine. You can do this using the WebSphere administrative console or **wsadmin**.
3. If your HTTP server is on a remote system, copy the plug-in configuration file to the remote Web server machine. The file is located in the IFS at *WAS_instance_dir/config/cells/plugin-cfg.xml*.

This is the plugin-cfg.xml entry when the WebSphere file serving feature is enabled:

```
<UriGroup Name="default_host_server1_apprsvl1_Cluster_URIs">
...
<Uri AffinityCookie="JSESSIONID" Name="/itsobank/*"/>
</UriGroup>
```

WebSphere serves all URIs starting with */itsobank/...*

Now that you disabled the file-serving feature of WebSphere, only JSP and servlet URIs are served by the application server. Regenerating the plug-in configuration updates the plugin-cfg.xml entry as follows:

```
<UriGroup Name="default_host_server1_apprsvl1_Cluster_URIs">
...
<Uri AffinityCookie="JSESSIONID" Name="/itsobank/servlets/Transfer"/>
<Uri AffinityCookie="JSESSIONID" Name="/itsobank/servlets/JMSTransfer"/>
<Uri AffinityCookie="JSESSIONID" Name="/itsobank/*.jsp"/>
<Uri AffinityCookie="JSESSIONID" Name="/itsobank/*.jsw"/>
<Uri AffinityCookie="JSESSIONID" Name="/itsobank/*.jsw"/>
<Uri AffinityCookie="JSESSIONID" Name="/itsobank/j_security_check"/>
<Uri AffinityCookie="JSESSIONID" Name="/itsobank/servlet/*"/>
</UriGroup>
```

WebSphere has intelligently updated the plug-in file to let the Web server serve static content. The Web server passes dynamic URIs for servlets and JSPs back to WebSphere.

4. Set up the Web server to serve the application's static content:
 - a. Extract the static content from the application archive and copy it to the Web server documents folder. In our example, the name of the folder is */www/htdocs/itsobank*.
To prevent errors and to make the process repeatable, it is best to use some type of script to extract and deploy the static content to the Web server. Alternatively, intelligent content management software can be used to automatically route documents around the network when they are updated.

- b. Add an alias for the application's static content folder to the IBM HTTP Server configuration:

Open your server's configuration file in IBM Web Administration for iSeries. Add the required alias to the end of the file, for example:

```
Alias /itsobank/ "/www/htdocs/itsobank"
```

- c. Save your changes.
 - d. Restart the IBM HTTP Server.
5. Try running the application from your browser.

The Web server should serve the static content. If you stop the application server, you should still see the static pages from the Web server, but you cannot access servlets or JSPs.

For more information, see 9.5.12, "Consideration with static content" on page 315.

6.7 References

Here are some other resources for you to read and learn more about how to improve the performance of your HTTP Server (powered by Apache) Web server:

- ▶ **iSeries Information Center**

The iSeries Information Center is a good starting point for performance-related topics that include the logging of information with iSeries Collection Services:

<http://publib.boulder.ibm.com/series/v5r2/ic2924/info/rzahx/rzahxebushttp.htm>

- ▶ *HTTP Server (powered by Apache): An Integrated Solution for IBM @server iSeries Servers*, SG24-6716

This IBM Redbook provides details about the HTTP Server.

- ▶ *iSeries Performance Capabilities Reference Version 5, Release 2*, SC41-0607

This is the definitive guide to performance on the iSeries server. This manual, which is updated regularly, has a good chapter on Web serving and communications performance.

Archived



Tuning the Java virtual machine

The first part of this chapter describes the Java virtual machine (JVM) architecture on iSeries. It discusses such JVM features as garbage collector, runtime framework, and heap allocation.

The second part of this chapter provides instructions for tuning JVM on the iSeries. It gives you the guidelines for choosing the optimal parameters for JVM on the iSeries server.

7.1 Java virtual machine on iSeries

Java, as a programming language, uses the famous concept: *write once, run anywhere*. The key part to support this concept is JVM. It acts as the isolation barrier between Java programs and the underlying operating system. Java programs are written to application programming interfaces (APIs) that don't depend on a specific operating system or hardware platform.

To support this environment, Java programs are compiled to the platform-independent *bytecode*. This bytecode can't be executed as is. It requires one additional step, *interpretation*. Interpreting the Java bytecode is one of the major functions of any JVM. As a result, an operating system provider or third-party vendor has to implement a JVM for a specific operating environment, such as OS/400.

7.1.1 High level view of JVM on the iSeries server

On the iSeries server, JVM is provided as part of OS/400 or i5/OS™. It is included in IBM Developer Kit for Java product (5722-JV1). JVM on iSeries is implemented in System Licensed Internal Code (SLIC), below the machine interface (MI). As such, it's better optimized than the JVMs on other platforms, where they run on top of an operating system.

Support for multiple Java Developer Kits

IBM Developer Kit for Java (5722-JV1) has several options. Each option corresponds to a different Java Developer Kit (JDK) version. JDK includes the Java interpreter, Java classes, and Java development tools: compiler, debugger, disassembler, appletviewer, stub file generator, and documentation generator.

Your iSeries server supports multiple JDKs simultaneously, but only through multiple JVMs. A single JVM runs one specific JDK. The `java.version` system property determines which JDK to run. When a JVM is up and running, changing the `java.version` system property has no effect. Table 7-1 shows the supported JDKs based on the release of OS/400 or i5/OS.

Table 7-1 Support for JDKs in V5R2 and V5R3

| OS/400 or i5/OS version | 5722JV1 option | JDK level |
|-------------------------|----------------|-----------|
| V5R2 | 3 | 1.2 |
| | 4 | 1.1.8 |
| | 5 | 1.3 |
| | 6 | 1.4 |
| V5R3 | 5 | 1.3 |
| | 6 | 1.4 |

Note: If several JDKs are installed, by default, the system uses the higher version first. For example, if option 5 and option 6 are installed, the system uses option 6 (JDK 1.4) first.

Starting a JVM

A JVM on iSeries starts when you perform one of the following steps:

- ▶ Use the `RUNJVA` or `JAVA CL` command
- ▶ Run the `java` command in Qshell
- ▶ Start a WebSphere Application Server instance

You may think of a JVM as a special type of operating system. It requires some time to prepare a runtime environment for execution of a Java program. It starts multiple threads, loads multiple Java classes, and so on.

Runtime environment

The Java programming language has built-in support for multithreading. This feature implies that JVM on iSeries has to run in a job that supports multiple threads, for example batch immediate (BCI).

Figure 7-1 shows a simplified view of JVM. It also shows a special type of a thread called the *garbage collector*. This thread is responsible for cleaning up a JVM heap.

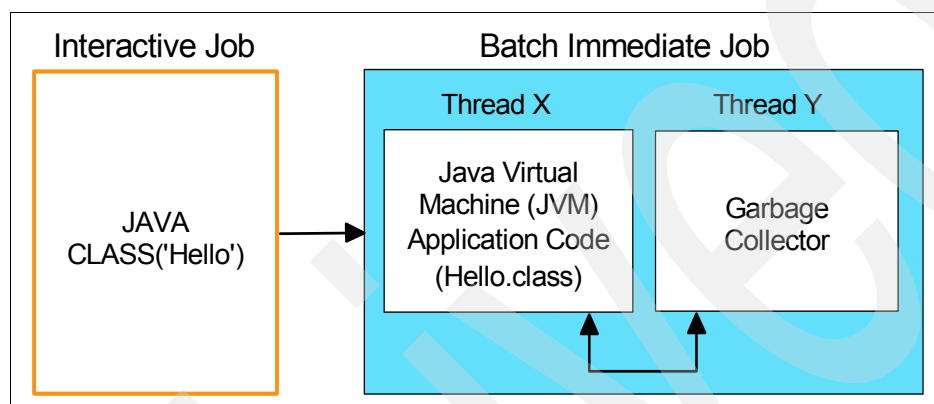


Figure 7-1 JVM environment on iSeries

A Java program creates many objects during its execution. These objects are created in a JVM heap. When an object is dead (meaning that no other object has a reference to this object), garbage collector removes this object from the heap. This function of the Java language and JVM is performed automatically.

There are two operating modes for the garbage collector:

- ▶ **Asynchronous:** In this mode, the garbage collector tries to clean up the heap without stopping other threads. It runs in the background.
- ▶ **Stop-and-copy:** In this mode, all active threads are suspended while the garbage collector cleans up the heap.

JVM on iSeries uses the asynchronous algorithm for the garbage collector. See 7.2, “Garbage collection in iSeries JVM” on page 223, for more details about garbage collector.

Important: JVM and Java programs create a high number of threads. Make sure that you have an adequate setting for the activity level in the subsystem where you run JVM.

Also, the health of JVM and garbage collector plays the most important role in ensuring good performance of your Java and WebSphere applications.

7.1.2 Runtime modes of execution

Java programs can be executed in different *execution modes*. An execution mode is the manner in which a JVM converts Java bytecode into the machine instructions. There are trade-offs between these modes.

First, this section describes each mode and then provides tips to help you choose a specific mode of execution for your Java program. You may select one of the following runtime modes of execution for your Java program:

- ▶ **Interpreted:** In this mode, JVM interprets (converts) each bytecode into machine instructions, one instruction at a time.
- ▶ **Direct processing:** In this mode, the entire method is converted into the machine instructions on the first call to the method.

Note: You can also use the Create Java Program (CRTJVAPGM) command to create a Java program object before you invoke it.

This converted version of the Java program or method is saved as a Java program object in OS/400. Then this Java program object is used for the subsequent executions of this method. One copy of the Java program object for a Java class is used per system.

- ▶ **Just-in-time compiler (JIT):** In this mode, the entire method is converted (compiled) into the machine instructions. The compiled version of the method is stored in the JIT heap. For all subsequent request to a method, JIT uses the compiled version from the heap. Learn more in “JIT” on page 222.
- ▶ **JIT compiler and direct processing:** This is the mixed mode. If a method has a corresponding Java program object, it runs using direct processing. For the methods that don't have a corresponding Java program object, they run under JIT.

Direct processing

A special component of OS/400, called *Java transformer*, is used to enable direct processing of the Java methods. The Java transformer creates an optimized program object that is persistent and is associated with the class file. In the default case, the program object contains a compiled, 64-bit RISC machine instruction version of the class. The Java interpreter does not interpret the optimized program object at runtime. Instead, it directly runs when the class file is loaded.

You can select an optimization level with Java transformer. Table 7-2 lists the available optimization levels along with information about the type of optimization and available debugging capabilities.

Table 7-2 Java transformer optimization levels

| Optimization level | Comments |
|--------------------|--|
| 0 | The Java program created does not contain machine-specific instructions. The Java program is interpreted from the bytecodes when it is started. Variables can be displayed and modified while debugging. |
| 10 | This is the least optimized level. The Java program contains a compiled version of the class file bytecodes and has only minimal additional compiler optimization. Variables can be displayed and modified while debugging. |
| 20 | The Java program contains a compiled version of the class file bytecodes and has some additional compiler optimization. Variables can be displayed but not modified while debugging. |
| 30 | The Java program contains a compiled version of the class file bytecodes and has more compiler optimization than optimization level 20. During a debug session, user variables cannot be changed, but can be displayed. The presented values may not be the current values of the variables. |

| Optimization level | Comments |
|--------------------|--|
| 40 | This is the highest optimized level. The Java program contains a compiled version of the class file bytecodes and has more compiler optimization than optimization level 30. All call and instruction tracing is disabled. |

To enable direct processing for your applications, choose one of the following methods. The method depends on how you invoke an application:

- Use the CRTJVAPGM command with the OPTIMIZE parameter, for example:

```
CRTJVAPGM CLSF('/tmp/myPackage/myApp.class') OPTIMIZE(40)
```

With this method, a Java program object is created *before* you execute this Java class. When you're ready to run this class, make sure that the java.compipler Java system property is set to jitc_de (the default value) or NONE. For example, add the following environment variable:

```
ADDENVVAR ENVVAR(JAVA_COMPILER) VALUE(jitc_de) LEVEL(*SYS)
```

Or use the command line argument if you run this class from Qshell:

```
java -Djava.compiler=jitc_de myPackage.myApp
```

You use CRTJVAPGM when you want to create a Java program object or objects before you run your application.

- If you use the Run Java (RUNJVA) command, specify the optimization level by using the OPTIMIZE parameter, for example:

```
RUNJVA CLASS('myPackage.myApp') OPTIMIZE(40)
```

In this case, a Java program object is created on the first touch to a method.

- If you invoke your application from the Qshell command line, then you need to specify two command line arguments:

```
java myPackage.myApp -Djava.compiler=NONE -Dos400.defineClass.optLevel=40
```

In this example, a Java program object is created for any Java class in this application on the first touch to the class. After that, any subsequent calls to this class run the compiled code in the corresponding Java program object.

- If you run your applications in WebSphere Application Server, add two custom properties to the JVM settings (see Figure 7-2):
 - **java.compiler:** Set this to NONE.
 - **os400.defineClass.optLevel:** Set this to a desired optimization level.

After you make these changes, restart your WebSphere Application Server instance.

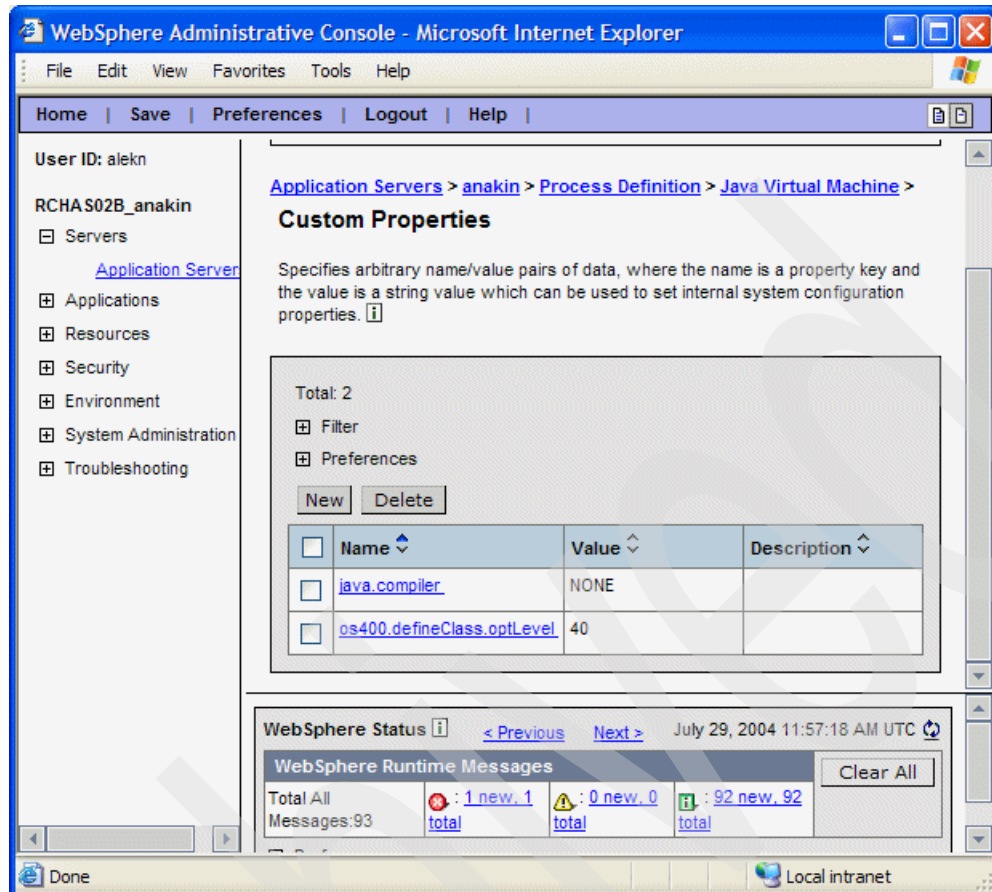


Figure 7-2 JVM custom properties

JIT

JIT is now a standard for running a Java program. The idea of JIT is simple. To optimize the performance of a Java application, any method is compiled into the machine instructions. The compiled code is cached in a JIT heap. For a subsequent execution of a method, JIT doesn't compile its code again. It uses the compiled version saved on the JIT heap.

This works fine for small Java programs that operate with few classes. But imagine if you run an enterprise application that uses thousands of the Java classes. For JIT to keep the compiled version of all classes on the heap, it has to use a very large heap. With limited system resources, this is unrealistic.

Therefore, JIT has to operate with a limited size heap. When the heap is full, JIT has to remove some compiled methods, for example *methodA* and *methodB*, from the heap to free some memory for new classes. When application needs to execute either *methodA* or *methodB* after that, the compiled version of the method is not on the heap. JIT has to compile this method again. This is why, until V5R2, JIT runtime mode was slower than direct execution, on average, with optimization level of 40.

JIT with Mixed Mode Interpretation (MMI)

The situation has changed for V5R2. A new technology called *Mixed Mode Interpretation* (MMI) was introduced with JIT V4 on the iSeries server.

If you look at the execution patterns of the Java classes in any application, you can easily identify a few classes and methods that are executed more frequently than others. Wouldn't it

be nice if you could make sure that the methods, that are executed more often, are always on the JIT heap? But how can you do this? The answer is simple: Use a *threshold* for each method.

The threshold helps to decide if a method should be compiled by the JIT. That is, the first n times, each method is run through the interpreter. On the $n+1$ invocation of a method, JIT compiles it and saves the compiled version of a method on the heap. With this algorithm, the methods that are executed less than n times are never run through the JIT. And there is a high chance that the method that is invoked the most is always on the JIT heap.

You can use the default values for JIT with MMI or you can modify them. See “Default settings” on page 223 for more information.

Direct processing or JIT: Performance impact

The startup time for a JVM running with the direct execution mode is faster than with JIT. However, when the application is warmed-up, the JVM running with JIT V4.0 shows on average 10% to 15% better performance than with the direct execution mode.

Default settings

The default runtime mode for a JVM on iSeries in V5R2 and later is *jite_de*. This means that:

- ▶ If a method has a corresponding Java program object, the direct execution mode is use.
- ▶ If a method doesn't have a corresponding Java program object, the JIT runtime mode is used.

The default value for MMI is 2000 invocations before a method is compiled by JIT. You can modify this value with the Java system property of *os400.jit.mmi.threshold*:

- ▶ A value of zero disables MMI and compiles methods when they are first called.
- ▶ Values lower than the default tend to both lengthen the startup time and degrade the ultimate performance.
- ▶ Values higher than the default initially degrade performance until they reach the threshold. Then they tend to improve ultimate runtime performance.

7.2 Garbage collection in iSeries JVM

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 the code to explicitly free (or delete) their objects that, otherwise, 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. When the object is collected (deleted), JVM automatically allocates the storage space for other objects.

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 control over 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 user's response time.

The iSeries has concurrent and asynchronous garbage collection, which results in consistent performance to the end user. Later this chapter explains how to tune garbage collection on the iSeries server to achieve the optimum performance.

7.2.1 Understanding garbage collection

Garbage collection has a price. Whether it is running as stop and copy or concurrent, it must still use central processing unit (CPU) resources to perform its task. Understanding how the garbage collector works can help you understand how to tune it.

Garbage collector execution on the iSeries (but not on other platforms) is based on the *initial heap size parameter*. 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 that the heap has grown since the last cycle (or since startup, if no cycle has yet been run). Figure 7-3 illustrates the iSeries JVM garbage collection cycles.

For example, as shown in Figure 7-3, if you specify an initial heap size of 96 MB, the first cycle runs after the 96 MB of space has been allocated. The garbage collector then frees some of that space for reuse (36 MB in the example in Figure 7-3). An additional 60 MB are added to the heap to make the size of the free heap equal to the initial heap size, which is 96 MB in our example. Then garbage collection waits until another 96 MB is allocated before running again.

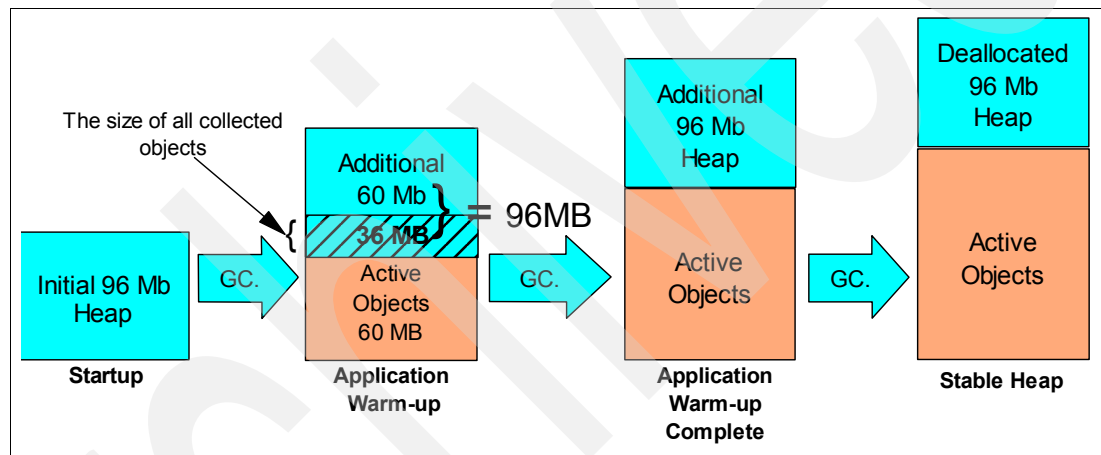


Figure 7-3 Garbage collection cycles example

Prior to V5R2, iSeries JVM did not perform heap compaction. Starting with V5R2, heap can be compacted.

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. 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.

7.2.2 Garbage collection performance tuning

To achieve the best WebSphere application performance on iSeries from the JVM perspective, you need to determine the optimal frequency of garbage collection cycles. The rule of thumb is that time spent on garbage collection should be less than 10% of the total CPU time. In addition, you must consider your system's resources, in particular CPU and memory. The initial heap size determines the frequency of garbage collections. When starting garbage collection performance analysis, start with the recommended values in Table 7-3.

Table 7-3 Starting values for the initial heap size parameter

| Processors | Initial heap size |
|------------|-------------------|
| 1 to 2 | 96 MB |
| 4 to 8 | 256 MB |
| 12 or more | 768 MB |

Tip: Another rule of thumb is to set the initial heap size for a three-way and above system to 64 MB per processor, and then increase this value as needed.

These recommendations are based on the assumption that your system has enough main storage to handle larger heap sizes. After you choose a reasonable starting point, start your application and let it run for a while under the maximum load that you intend to handle. Give it time to reach a steady state. A few minutes is usually sufficient. It's best to use a load-generation tool to put a constant load on your system. This allows you to tune your application in a development environment rather than a production environment. In addition to providing a constant load (allowing you to see the effects of changes more accurately), this allows you to make changes as necessary without affecting users.

In systems with limited memory, it may be necessary to set the garbage collection threshold to a lower value to increase collection frequency and decrease heap size, allowing the entire heap to be kept in memory. Use the Work with System Status (WRKSYSSTS) command or Collection Services to monitor the non-database paging and faulting rates. If these rates become too high, the heap value may be too large. The definition of "too large" depends on a variety of factors, such as system size, number of disks, and system workload. In general, sustained non-database paging rates that are greater than 10 faults per second by Java programs is generally cause for concern. Higher paging rates are acceptable during *warm-up* periods.

High paging rates may result from having the garbage collection threshold set too high or may be a symptom of a larger problem. In this case, the first step should be to isolate the JVM in its own memory pool. This reduces the effects that other applications may have on the JVM. It also makes it easier to identify whether the problem is with the garbage collection settings, system configuration, or simply not enough hardware to handle the workload.

In cases where memory is especially limited, it may be useful to set the maximum heap size. Normally, you should leave this at the default value *NOMAX, which means that garbage collection runs only when the garbage collection threshold is reached. If a maximum heap size is set, the collector runs whenever the heap reaches that maximum size.

Unlike a normal garbage collection, if the maximum size is reached, all application threads must wait until the collector finishes before they can continue running. This results in undesirable pause times. Therefore, it's preferred to use the maximum heap size as a safety net to handle times of unexpected heap growth and to ensure that the heap doesn't grow larger than the available memory. The garbage collection threshold should be set so that this maximum size is never actually reached under normal circumstances.

Setting the initial heap size in WebSphere

This section takes you through the process of analyzing garbage collection performance and changing initial heap value to achieve the optimal performance. The best way to determine the optimal heap size is to run a set of experiments with different initial heap size values. The goal in this section is to collect some data that will help to determine whether WebSphere

Application Server is tuned for garbage collection. In particular, we are interested in the following indicators:

- ▶ Garbage collection CPU usage
- ▶ Garbage collection cycle time
- ▶ Heap size
- ▶ Non-database paging

There are multiple ways to collect this information. In this case, we are using tools that are available in i5/OS or WebSphere Application Server. First, we modify JVM settings in WebSphere Application Server.

1. Open the WebSphere administrative console.
2. In the navigation tree, expand **Servers** and click **Application Servers**.
3. Select the application server you want to tune.
4. In the Configuration tab of the selected application server, under Additional Properties, click **Process Definition**.
5. In the Process Definition window, click the **Java Virtual Machine** link (see Figure 7-4).

The screenshot shows the 'Configuration' tab of the WebSphere administrative console. The 'General Properties' section contains three rows: 'Executable name' with an empty text box, 'Executable arguments' with an empty text box and a scroll bar, and 'Working directory' with a text box containing '\$(USER_INSTALL_ROOT)'. To the right of these fields are informational icons and descriptions. Below the 'General Properties' section is the 'Additional Properties' section, which is expanded to show a list of links: 'Java Virtual Machine' (highlighted with a red circle), 'Process Execution', 'Process Logs', 'Environment Entries', 'Monitoring Policy', and 'Logging and Tracing'. Each link has a corresponding description.

Figure 7-4 Selecting the Java Virtual Machine link

6. Find the Initial Heap Size field. In this case, we test on a two-way system. Following the recommended guidelines, we set the initial heap size to 96 MB as shown in Figure 7-5.

The screenshot shows the 'Java Virtual Machine' settings window. It contains two rows: 'Initial Heap Size' with a text box containing '96' and 'Maximum Heap Size' with a text box containing '0'. To the right of these fields are informational icons and descriptions. The 'Initial Heap Size' description states: 'Specifies the initial heap size available to the JVM (in megabytes)'. The 'Maximum Heap Size' description states: 'Specifies the maximum heap size available to the JVM, in megabytes. The default is 256.'

Figure 7-5 Setting the initial heap size

7. Select the **Verbose garbage collection** check box as shown in Figure 7-6. This option allows you to collect important data about garbage collection. Use this option only during debugging.

| | | |
|----------------------------|-------------------------------------|---|
| Verbose garbage collection | <input checked="" type="checkbox"/> | <small>i Specifies whether to use verbose debug output for garbage collection. The default is not to enable verbose garbage collection.</small> |
|----------------------------|-------------------------------------|---|

Figure 7-6 Enabling the verbose garbage collection

8. Click the **Apply** button.
9. Click the **Save** configuration link as shown in Figure 7-7.



| |
|---|
| <p>Message(s)</p> <p> Changes have been made to your local configuration. Click Save to apply changes to the master configuration.</p> <p> The server may need to be restarted for these changes to take effect.</p> |
|---|

Figure 7-7 Saving the changes

10. Restart WebSphere Application server.

Using PEX to collect performance data

Next, create a Performance Explorer (PEX) trace that shows CPU utilization for garbage collection. Complete the following steps to create PEX trace:

1. Start a 5250 session.
2. In an iSeries command prompt, create a PEX trace definition:


```
ADDPEXDFN DFN(GCTEST) TYPE(*TRACE) JOB((*ALL)) MAXSTG(1000000) INTERVAL(1)
TRCTYPE(*SLTEVT) SLTEVT(*YES) BASEVT((*PMCO)) TEXT('TPROF with run cycle sampling
interval of 1 ms')
```
3. Start the trace:
 - a. Type the STRPEX command and press Enter.
 - b. Enter a session ID (unique name), for example GCTEST64MB. For Option, select ***NEW**. Press Enter.
 - c. In the Definition field, select ***SELECT** and press Enter.
 - d. Type the trace definition you created (GCTEST). Notice the message that indicates that PEX trace has started.
4. Start your application or load tool. In our case, we started our OpenSTA test.
5. When your test is finished, on an OS/400 command line, type ENDPEX and press Enter.
6. Select the trace (GCTEST), and press Enter.
7. Print a PEX Report by running the following command:


```
PRTPEXRPT MBR(GCTEST64MB) LIB(QPEXDATA) TYPE(*PROFILE)
```
8. Type the WRKSPLF command and look for the QVPERPT spooled file. Type option 5 (Display) next to the report. If you have multiple QVPERPT files, verify the member name, which should be equal to your session ID, which in our case is GCTEST64MB.

9. In the PEX trace report, search for the JAVAGC indicator as shown in Figure 7-8. In our first test garbage collection, CPU utilization is 12.9%. The best value for garbage collection CPU utilization is around 3%, but values under 10% are also acceptable.

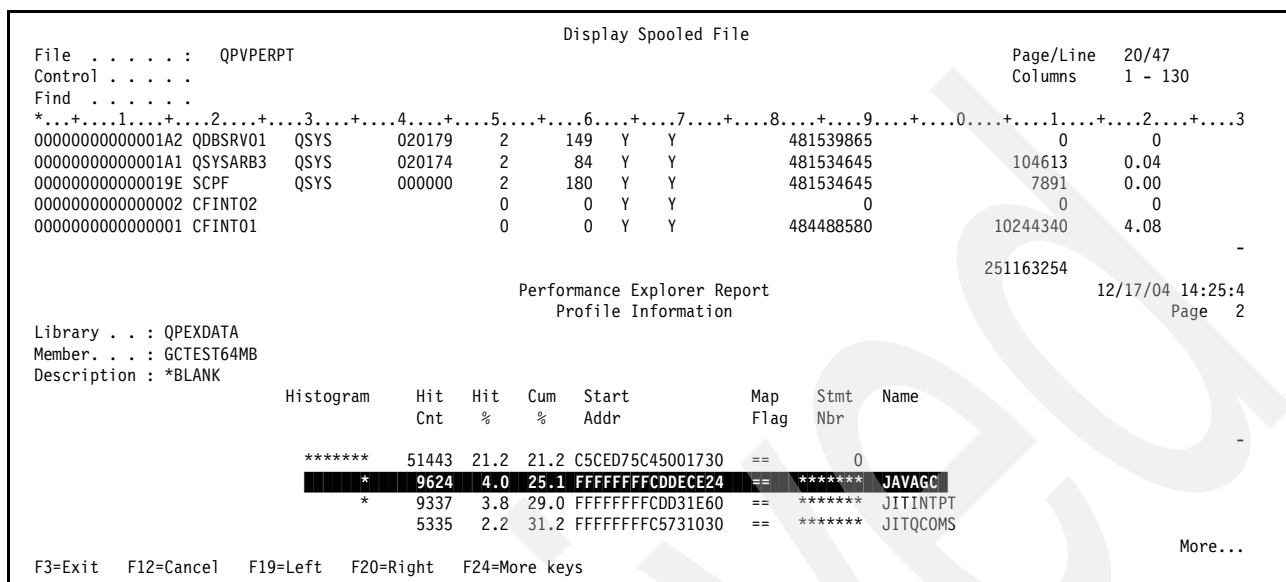


Figure 7-8 Performance report

Analyzing the verboseGC data

Let's look at the verboseGC output. We turned on verboseGC trace in the WebSphere Application Server JVM settings. Using WebSphere Application Server V5.1, you can find verboseGC output in /QIBM/UserData/WebAS51/Base/<instance name>/logs/<server name>/native_stdout.log. Example 7-1 shows a sample of the verboseGC output.

Example 7-1 verboseGC output

```
GC 21: live objects 660435; collected objects 1322925; collected(KB) 112919.
GC 21: queued for finalization 0; total soft references 1835; cleared soft references 0.
GC 21: current heap(KB) 286816; current threshold(KB) 98304.
GC 21: collect (milliseconds) 5116.
GC 21: current cycle allocation(KB) 19920; previous cycle allocation(KB) 98334.
GC 21: total weak references 645; cleared weak references 23.
GC 21: total final references 2501; cleared final references 1001.
GC 21: total phantom references 4; cleared phantom references 0.
GC 21: total JNI global weak references 0; cleared JNI global weak references 0.
```

The following list explains some of the parameters in the verboseGC output:

- ▶ **GC 21:** The twenty-first garbage collection cycle since JVM started
- ▶ **Live objects:** Number of objects currently active in the JVM
- ▶ **Collected objects:** Number of objects collected during this cycle
- ▶ **Collected (KB):** Total size of the objects collected during this cycle
- ▶ **Current heap (KB):** The current heap size
- ▶ **Current threshold (KB):** The threshold value
- ▶ **Collect (milliseconds):** Elapsed time for this cycle
- ▶ **Current cycle allocation (KB):** Memory allocated since the current cycle began
- ▶ **Previous cycle allocation (KB):** Memory allocated since the last cycle began

The current threshold is the value set for the initial heap size (96 MB in the example output). The previous cycle allocation is normally close to this value, because the garbage collection

cycle is triggered when the amount of memory allocated since the last cycle began reaches the threshold value. The example output shows that the garbage collection cycle took more than 5 seconds to complete. During that time, the current cycle allocation reached more than 19 MB. This is about 21% of the threshold value, which suggests that the total time between the beginning of this cycle and the next cycle is around 24 seconds. This cycle collected about 1.3 million objects (collected objects), leaving only 660 K objects in the heap at the end of the cycle (live objects).

In general, it's best to have a low cycle time. One to two seconds is ideal, but times of five to 10 seconds are common for WebSphere Application Server applications. It's also best to have some time between collection cycles. That is current cycle allocation should be less than the current threshold. These two goals work against each other. The increasing threshold value allows the heap to grow, resulting in more time between cycles, but lengthens each cycle. Decreasing the threshold shortens each cycle, but also shortens the time between cycles.

The key to tuning garbage collection is to find a balance between these two goals. This is why examining the CPU consumed by garbage collection is generally better than looking at values such as the current heap size. If you don't have the Performance Tools licensed program product (PT1 LPP) installed on your system, you may want to experiment with the threshold value. Use your application's throughput or response time as well as verbose garbage collection output to determine how to further tune the threshold.

Even if you have PT1, it may be useful to examine verbose garbage collection output to understand how your application uses the heap and to watch for changes in the garbage collection behavior as application and system loads change.

Verbose garbage collection output also shows heap size growth. After the heap has grown past available memory in the pool, paging occurs. A rule of thumb is that, if you have paging higher than 100 pages/second in the pool with your JVM, this paging can seriously inhibit the garbage collection's ability to manage the heap and collect objects. In this scenario, the paging itself can be the cause of further heap growth. As shown in Figure 7-9 during our test run, we reached the point where heap grew higher than the available memory amount.

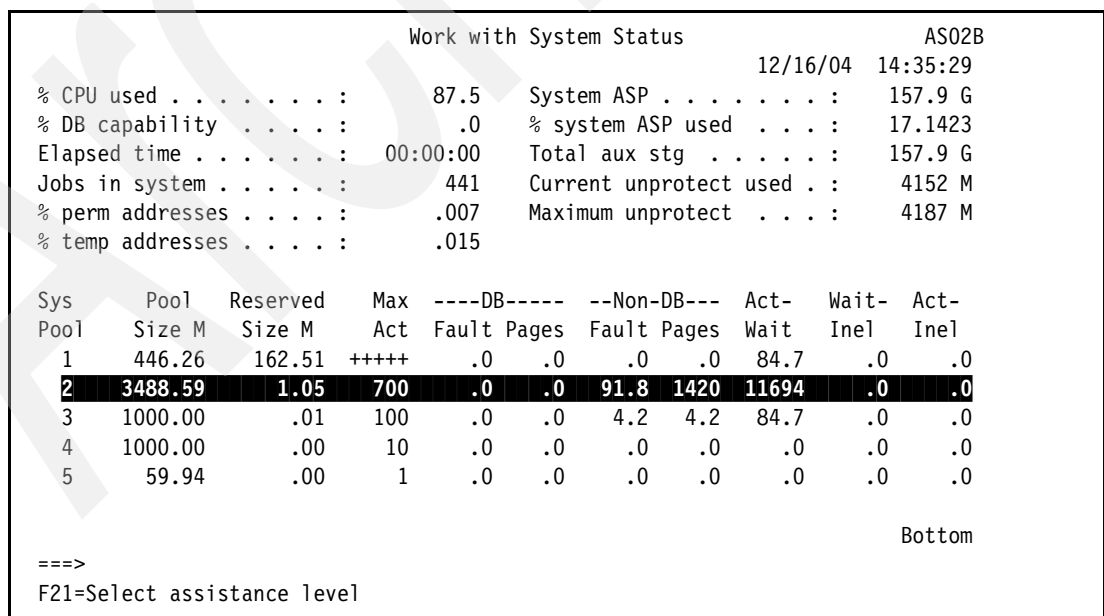


Figure 7-9 Monitoring the page fault rate

The evaluation of performance indicators in our test case showed that our initial heap size is too small. We adjusted the initial heap size to 256 MB in WebSphere Application JVM settings, restarted WebSphere Application Server, and reran the test case. The test results showed smaller CPU utilization, less paging than we observed by running WRKSYSSTS command, and higher garbage collection cycle time than we found in verbose garbage collection output.

We ran the test one more time with initial heap size set to 1024 KB. Our CPU utilization was at 1.5%, and we saw a slight increase in garbage collection cycle time. In our case, this is the optimal setting because CPU utilization is low, and garbage collection cycle time is acceptable. In addition, we saw higher throughput with the initial heap size is set to 1024 KB.

7.2.3 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 retains the object reference for the life of the application, the object is never garbage collected. Good coding practices can prevent such a scenario from occurring.

Another typical example is when a reference to an object is placed in a object structure, such as a vector or hash table. A vector or hash table object's life span is usually the same as the life span of a JVM where they run. Any reference of an object placed in a vector, for example, has to be explicitly removed.

To catch these types of memory leaks, you can use one of the available tools, such as WebSphere Development Studio Client for iSeries Advanced Edition or Performance Trace Data Visualizer (PTDV). Application Profiler, which is part of WebSphere Development Studio Client, is a sophisticated tool. It allows you to collect a lot of information about the execution of your Java program. It's suitable for memory leak detection.

Tuning WebSphere Application Server Base

This chapter covers some of the more important tuning parameters of the WebSphere Application Server. Each application server instance has several parameters that can influence application performance. You can use the WebSphere Application Server administrative console to configure and tune applications, Web containers, Enterprise JavaBean (EJB) containers, application servers, and nodes in the administrative domain. You can also configure settings for other components of your iSeries environment to optimize performance.

Note: The information in this chapter is based on WebSphere Application Server *Version 5.1.0.4*

8.1 Adjusting WebSphere Application Server system queues

WebSphere Application Server contains interrelated components that must be tuned to support the custom needs of your On Demand Business application. These adjustments help the system achieve maximum throughput while maintaining the overall stability of the system.

This group of interconnected components is known as a *queuing network*. These queues or components include the network, Web server, Web container, EJB container, data source, and possibly a connection manager to a custom back-end system. Each of these resources represents a queue of requests waiting to use that resource. The various queue settings include:

- ▶ IBM HTTP Server: See Chapter 6, “Tuning the IBM HTTP Server (powered by Apache)” on page 199.

The `ThreadsPerChild` parameter is the main setting we adjust. This directive specifies the maximum number of concurrent client requests that the server processes at any time. The Web server uses one thread for each request that it processes.

- ▶ Web container: See 8.2.1, “Web container” on page 239.

Each application server includes a Web container. The application server routes servlet requests along a transport queue between the Web server plug-in and the Web container. The three main settings are Thread pool maximum size, HTTP transports `MaxKeepAliveConnections`, and `MaxKeepAliveRequests`.

- ▶ Object Request Broker: See 8.2.6, “Object Request Broker” on page 266.

An Object Request Broker (ORB) uses the Internet InterORB Protocol (IIOP) to manage the interaction between clients and servers. The Thread pool maximum size is the main setting that you adjust.

- ▶ Data source: See 8.2.4, “Data sources” on page 258.

Applications use data sources to access databases. The two main settings are Connection pooling and Statement cache size.

- ▶ Java Messaging Service (JMS): See 8.3, “Java Messaging Service tuning parameters” on page 278.

WebSphere Application Server supports asynchronous messaging as a method of communication based on the JMS programming interface. JMS provides a common way for Java programs (clients and J2EE applications) to create, send, receive, and read asynchronous requests, as JMS messages.

The four main settings are Listener service thread pool maximum size, Listener port maximum sessions, Listener port maximum messages, and Connection pooling.

Figure 8-1 shows an example of a WebSphere queuing network. Most of the queues that make up the queuing network are closed queues. A *closed queue* places a limit on the maximum number of requests present in the queue, while an *open queue* has no limit. A closed queue supports strict management of system resources. For example, the Datasource pool setting controls the size of the datasource queue. If the average connection occupies 3 MB of memory, a value of 30 connections for the connection pool limits the memory consumed by the data source to 90 MB.

In a closed queue, requests can be active or waiting. An *active request* is doing work or waiting for a response from a downstream queue. For example, an active request in the Web server is doing work, such as retrieving static HTML, or waiting for a request to be processed in the Web container. A *waiting request* is waiting to become active. The request remains in the waiting state until one of the active requests leaves the queue.

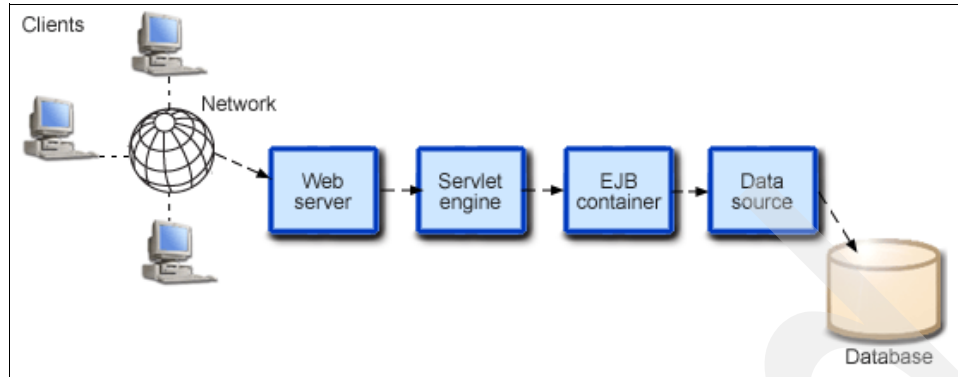


Figure 8-1 WebSphere queuing network example

8.1.1 Queue configuration tips

All Web servers supported by WebSphere Application Server are closed queues, as are WebSphere Application Server data sources. You can configure Web containers as open or closed queues. In general, we recommend that you use closed queues. EJB containers are open queues. If no threads are available in the pool, a new thread is created for the duration of the request.

If enterprise beans are called by servlets, the Web container limits the number of total concurrent requests into an EJB container, because the Web container also has a limit. The Web container limits the number of total concurrent requests only if enterprise beans are called from the servlet thread of execution. Nothing prevents a servlet from creating threads and overloading the EJB container with requests. Therefore, servlets should not create their own work threads.

The following sections provide tips to help you tune the queuing network configuration.

Minimizing the number of requests in WebSphere Application Server queues

Performance is usually improved if requests wait in the network, in front of the Web server, rather than waiting in the application server. That is, only requests that can be processed enter the queuing network. To achieve this result, set the size of upstream (closest to the client) queues to large, and specify progressively smaller sizes for downstream (further from the client) queues. Figure 8-2 provides an example of this configuration.

Queues in the queuing network become progressively smaller as work flows downstream. In this example, 200 client requests arrive at the Web server. Of those requests, 125 requests remain queued in the network because the Web server is set to handle 75 concurrent clients. As the 75 requests pass from the Web server to the Web container, 25 requests remain queued in the Web server and the remaining 50 are handled by the Web container. This process progresses through the data source until 25 user requests arrive at the final destination, the database server. Because there is work waiting to enter a component at each point upstream, no component in this system must wait for work to arrive. Most of the requests wait in the network, outside of WebSphere Application Server. This type of configuration adds stability, because no component is overloaded.

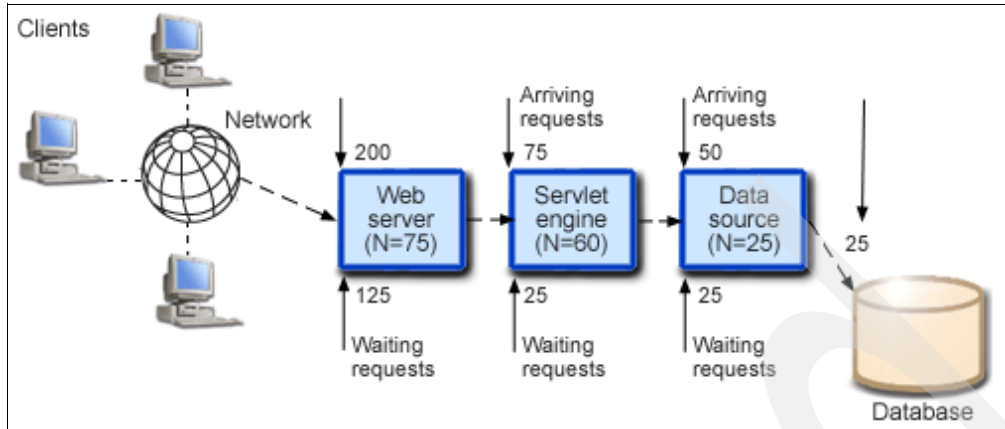


Figure 8-2 Upstream queuing network example

Drawing throughput curves

To run a test case that represents the full use of the production application, exercise all meaningful code paths or use the production application. Run a set of tests to determine when the system capabilities are fully stressed or when the network has reached the saturation point. At this point, you have the maximum throughput. Conduct these tests after most of the bottlenecks are removed from the application. The goal of these tests is to drive CPUs to near 100% utilization.

For maximum concurrency through the system, start the initial baseline experiment with large queues. For example, start the first experiment with a queue size of over 200 at each of the servers in the queuing network: Web server, Web container, and data source. Begin a series of experiments to plot a throughput curve, increasing the concurrent user load after each experiment. For example, perform experiments with 1, 2, 5, 10, 25, 50, 100, 150, and 200 users. After each test, record the throughput requests per second, and response times in seconds per request. As shown in Figure 8-3, the curve resulting from the baseline experiments resembles a typical throughput curve.

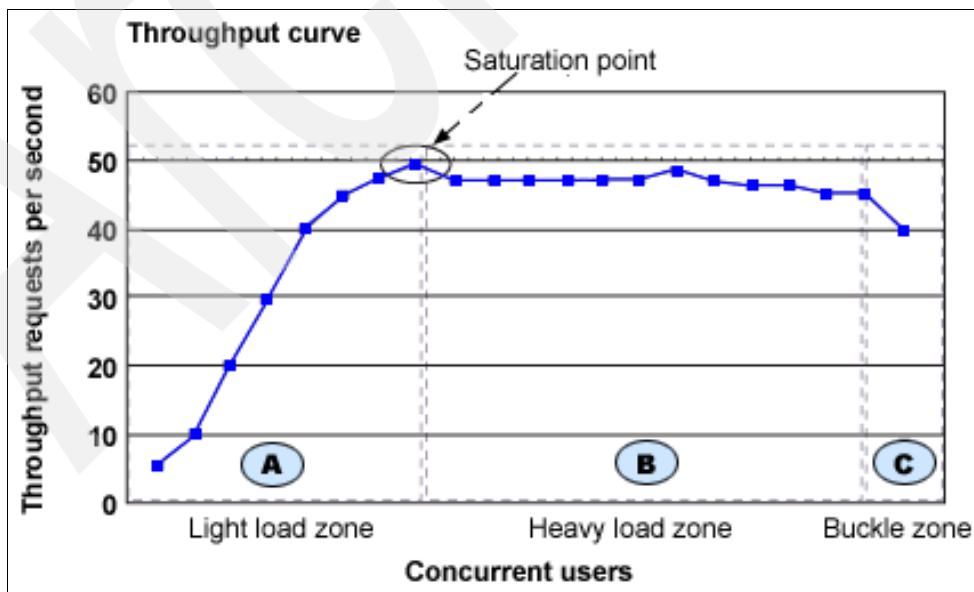


Figure 8-3 Throughput curve

The WebSphere Application Server throughput is a function of the number of concurrent requests present in the total system. Section A, the *light load zone*, shows that as the number of concurrent user requests increases, the throughput increases almost linearly with the number of requests. Under light loads, concurrent requests face little congestion within the WebSphere Application Server system queues. At some point, congestion starts to develop. Then throughput increases at a much lower rate until it reaches a saturation point that represents the maximum throughput value, as determined by some bottleneck in the WebSphere Application Server system. The most manageable type of bottleneck occurs when the WebSphere Application Server machine CPUs become fully utilized. To resolve this bottleneck, you must add processing power.

In Section B, the *heavy load zone*, as the concurrent client load increases, throughput remains relatively constant. However, the response time increases proportionally to the user load. That is, if the user load is doubled in the heavy load zone, the response time doubles. At some point, represented by Section C, the *buckle zone*, one of the system components becomes exhausted. At this point, throughput starts to decrease. For example, the system may enter the buckle zone when the network connections at the Web server exhaust the limits of the network adapter or if the requests exceed operating system limits for file handles.

If the saturation point is reached by driving CPU utilization close to 100%, you can move to the next step. If the saturation point occurs before system utilization reaches 100%, another bottleneck may be the cause. For example, the application may be creating Java objects and causing excessive garbage collection bottlenecks in the Java code.

There are two ways to manage application bottlenecks: remove the bottleneck or clone the bottleneck. The best way to manage a bottleneck is to remove it. You can use a Java-based application profiler to examine overall object utilization. For a list of available tools, see Chapter 4, “Tools for determining performance problems” on page 27.

Decreasing queue sizes as requests move downstream from the client

The number of concurrent users at the throughput saturation point represents the maximum concurrency of the application. For example, if the application saturates WebSphere Application Server at 50 users, using 48 users may produce the best combination of throughput and response time. This value is called the *Max Application Concurrency value*.

Max Application Concurrency becomes the preferred value for adjusting the WebSphere Application Server system queues. Remember that it is desirable for most users to wait in the network. Therefore, queue sizes should decrease when moving downstream farther from the client. For example, given a Max Application Concurrency value of 48, start with system queues at the following values: Web server 75, Web container 50, and Data source 45. Perform a set of additional tests with slightly higher and lower values to find the best settings.

To determine the number of concurrent users, view the Servlet Engine Thread Pool and Concurrently Active Threads metrics in the Tivoli Performance Viewer (see Figure 8-4).

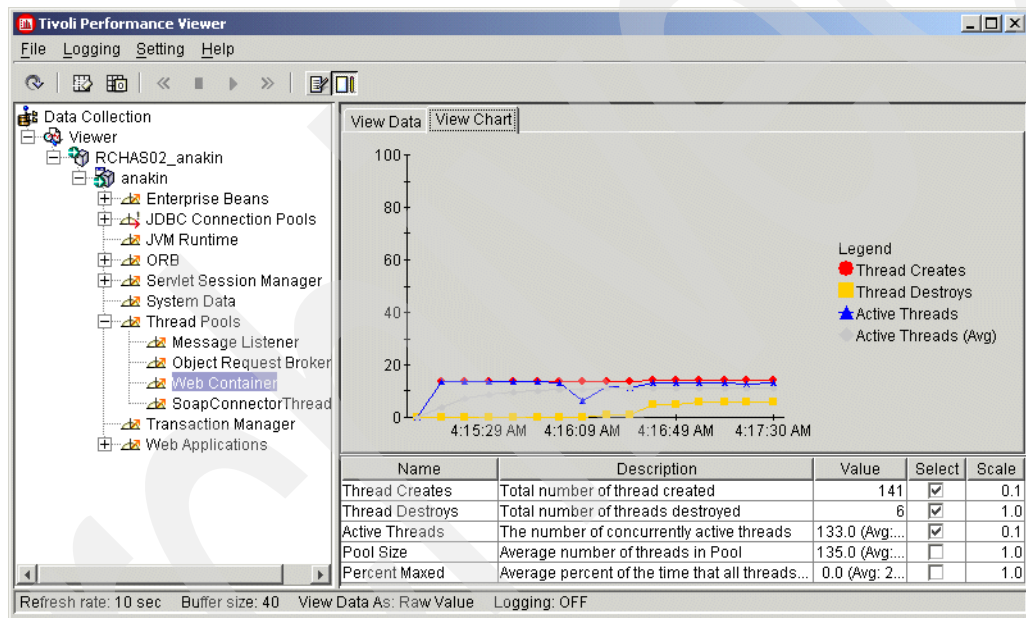


Figure 8-4 Number of concurrently active threads

Adjusting queue settings to correspond to access patterns

In many cases, only a fraction of the requests that pass through one queue enters the next queue downstream. For example, on a Web site with many static pages, a number of requests are fulfilled at the Web server and are not passed to the Web container. In this case, the Web server queue can be significantly larger than the Web container queue. In the previous example, the Web server queue was set to 75, rather than closer to the value of Max Application Concurrency. You can make similar adjustments when different components have different execution patterns.

Consider an application that spends 90% of its time in a complex servlet and only 10% of its time making a short Java Database Connectivity (JDBC) query. On average, 10% of the servlets are using database connections at any time. Therefore, the database connection queue can be significantly smaller than the Web container queue. Conversely, if the majority of servlet execution time is spent making a complex query to a database, consider increasing the queue values at both the Web container and the data source. Always monitor the CPU and memory utilization for both the WebSphere Application Server and the database servers to verify that the CPU or memory are not overloaded.

8.1.2 Enterprise bean method invocation queuing

Method invocations to enterprise beans are queued only for remote clients that make the method call. An example of a remote client is an enterprise bean client running in a separate Java virtual machine (JVM) (another address space) from the enterprise bean. In contrast, no queuing occurs if the enterprise bean client, either a servlet or another enterprise bean, is installed in the same JVM on which the enterprise bean method runs and on the same thread of execution as the enterprise bean client.

Remote enterprise beans communicate with the Remote Method Invocation over an Internet Inter-Orb Protocol (RMI-IIOP). Method invocations initiated over RMI-IIOP are processed by a server-side ORB. The thread pool acts as a queue for incoming requests. However, if a remote method request is issued and there are no more available threads in the thread pool, a new thread is created. After the method request is completed, the thread is destroyed. Therefore, when the ORB processes remote method requests, the EJB container is an open queue due to the use of unbounded threads. Figure 8-5 illustrates the two queuing options for enterprise beans.

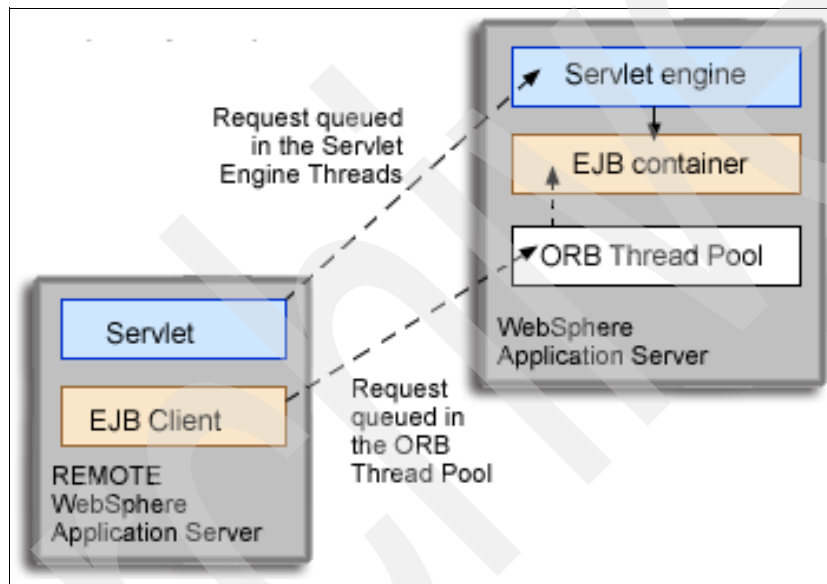


Figure 8-5 EJB queuing example

Analyzing the calling patterns of the enterprise bean client

When you configure the thread pool, it is important to understand the calling patterns of the enterprise bean client. If a servlet is making a small number of calls to remote enterprise beans and each method call is relatively brief, consider setting the number of threads in the ORB thread pool to a value lower than the Web container thread pool size value.

The degree to which the ORB thread pool value needs to be increased is a function of the number of clients (or servlets) that simultaneously call enterprise beans. It is also a function of the duration of each method call. If the method calls are longer or the applications spend a lot of time in the ORB, consider making the ORB thread pool size equal to the Web container size. If the servlet makes only short-lived or quick calls to the ORB, servlets can potentially reuse the same ORB thread. See Figure 8-6. In this case, the ORB thread pool can be as small as one-half of the thread pool size setting for the Web container.

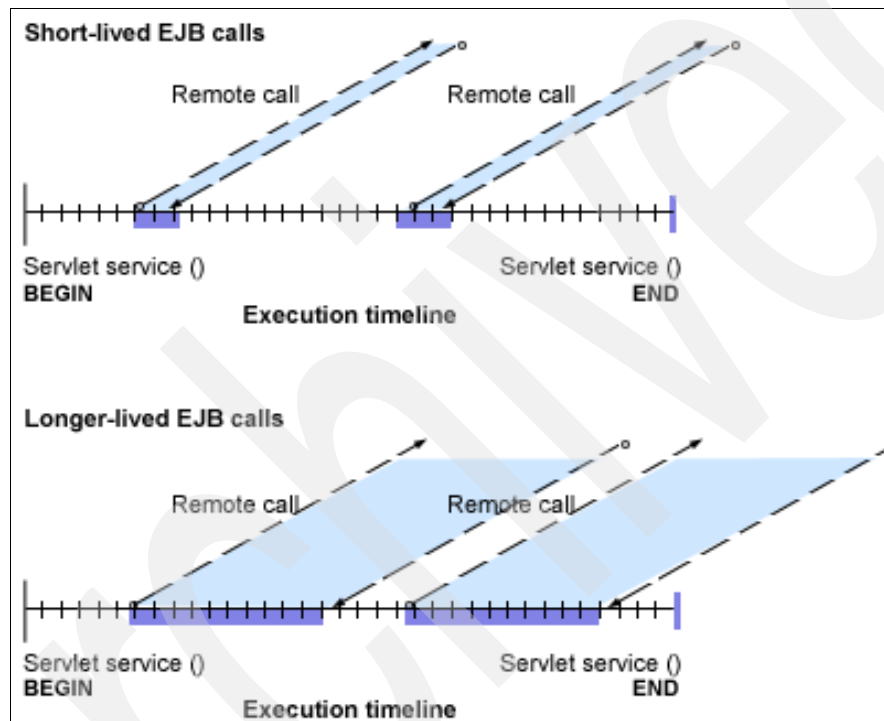


Figure 8-6 EJB calls

8.2 WebSphere Application Server tuning parameters

Each application server instance has several parameters that can influence application performance. You can use the WebSphere Application Server administrative console to configure and tune applications, Web containers, EJB containers, application servers, and nodes in the administrative domain. You can tune application server settings to control how an application server provides services for running applications and their components.

You can tune the following application server settings:

- ▶ Web container
- ▶ Session management
- ▶ Dynamic cache service
- ▶ Data sources
- ▶ EJB container

- ▶ Object Request Broker
- ▶ Process Priority

8.2.1 Web container

Each application server includes a Web container. The application server routes servlet requests along a transport queue between the Web server plug-in and the Web container. Default Web container properties are set for simple Web applications. However, these values may not be appropriate for more complex Web applications. You can adjust these parameters to tune the Web container based on the specific needs of your environment:

- ▶ Thread pool maximum size
- ▶ Growable thread pool
- ▶ MaxKeepAliveConnections
- ▶ MaxKeepAliveRequests

Thread pool maximum size

This value limits the number of clients that your application server can process concurrently.

Note: The information on the Settings page applies to all thread pools in WebSphere Application Server.

Viewing or setting Thread pool maximum size

To change or set this WebSphere Application Server value, follow these steps:

1. Start the administrative console.
2. In the topology tree, expand **Servers** and click **Application Servers**.
3. Click the name of the application server that you want to configure.
4. Click **Web Container**.
5. On the Web Container page, click **Thread Pool**.
6. Under General Properties, in the Maximum Size field, specify a value. See Figure 8-7.

[Application Servers](#) > [anakin](#) > [Web Container](#) >

Thread Pool

A thread pool allows components of the server to reuse threads to eliminate the need to create new threads at runtime. Creating new threads is typically a time and resource intensive operation. [i](#)

| Configuration | | | |
|----------------------------|-------------------------------------|---|--|
| General Properties | | | |
| Minimum Size | * | <input type="text" value="50"/> | threads i Specifies the minimum number of threads to allow in the pool. |
| Maximum Size | * | <input type="text" value="60"/> | threads i Specifies the maximum number of threads to allow in the pool. |
| Thread inactivity timeout: | * | <input type="text" value="3500"/> | milliseconds i Specifies the number of milliseconds of inactivity that should elapse before a thread is reclaimed. |
| Growable thread pool: | <input checked="" type="checkbox"/> | Allow thread allocation beyond maximum thread size i Specifies whether the number of threads can increase beyond the maximum size configured for the thread pool. | |

Apply OK Reset Cancel

Figure 8-7 Web container thread pool

7. Click **Apply** or **OK**.
8. Save the configuration.
9. Restart the application server.

Default value

The default value for the Maximum size field is 50.

Recommended value

Set this value to handle the peak load on your application server. We recommend that you specify a maximum size less than or equal to the number of threads processing requests in your HTTP server. A value in the range 25 to 50 is generally a good starting point. You can use the Tivoli Performance Viewer to monitor the number of threads being used.

Thread pool test

Tivoli Performance Viewer displays a metric called *Percent Maxed* that determines the amount of time that the configured threads are all in use. Percent Maxed has two values: the current and average value. If the current value is consistently 100%, the Web container can be a bottleneck, so you should increase the number of threads. See Figure 8-8.

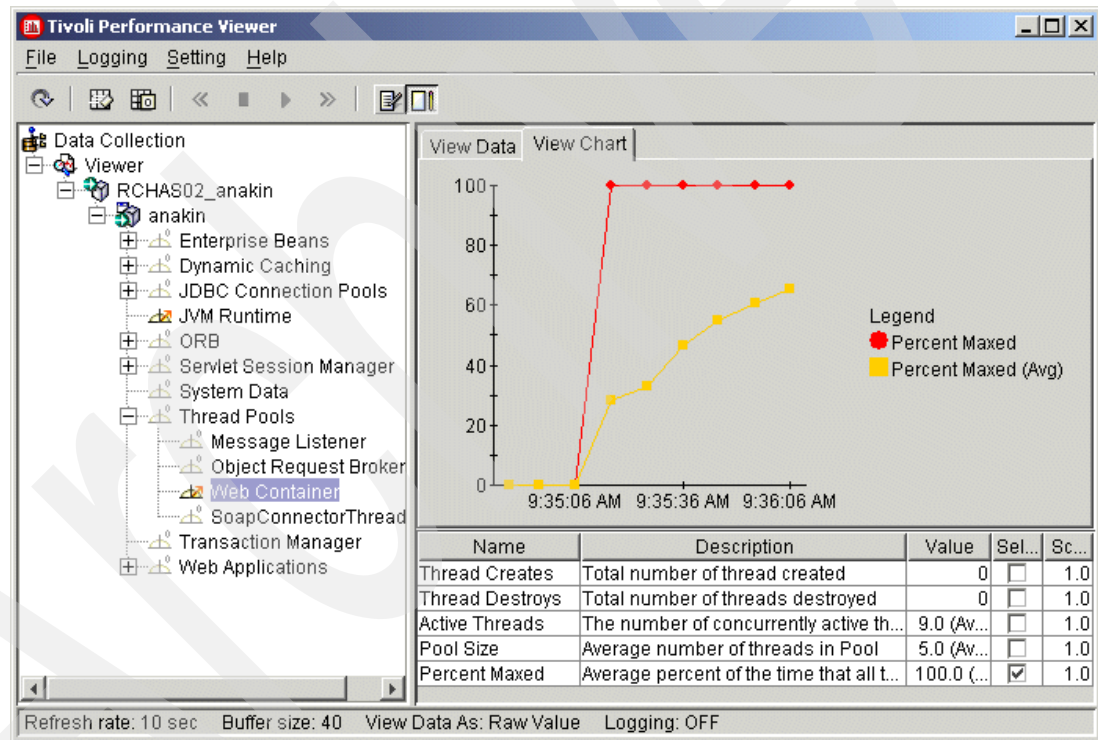


Figure 8-8 Percentage maxed

Table 8-1 shows a comparison between Thread Pool Size and Average Request Latency. The conclusion is, that not even a larger pool size results in a good response time or throughput.

Table 8-1 Thread pool test with 100 clients

| | First test | Second test | Third test | Fourth test |
|------------------------------|------------|-------------|------------|-------------|
| Minimum Size | 1 | 1 | 1 | 1 |
| Maximum Size | 50 | 100 | 100 | 150 |
| Growable thread pool | No | No | Yes | No |
| HTTP Requests | 12867 | 15851 | 15665 | 15866 |
| Average Request Latency (ms) | 1.568 | 1.234 | 1.464 | 1.564 |

Growable thread pool

This setting specifies whether the number of threads can increase beyond the maximum size configured for the thread pool.

Specifying Growable thread pool

To change or set this WebSphere Application Server value, use these steps:

1. Start the administrative console.
2. In the topology tree, expand **Servers** and click **Application Servers**.
3. Click the name of the application server that you want to configure.
4. Click **Web Container**.
5. On the Web Container page, click **Thread Pool**.
6. Select **Allow thread allocation beyond maximum thread size**. See Figure 8-7 on page 239.
7. Click **Apply** or **OK**.
8. Save the configuration.
9. Restart the application server.

Default value

The default value for the Growable thread pool field is Disabled.

Recommended value

We recommend that you do not enable this property if you are confident that the maximum thread pool size is large enough to adequately process the peak load on your application server. Enable this property if you want to allow the thread pool to exceed the configured maximum pool size. This setting is beneficial if the application server receives an unexpected increase in requests or if the maximum pool size is set too low.

In this scenario, additional threads are created to handle the increased number of requests. These connections are destroyed when the number of requests returns to its typical level. However, enabling the growable thread pool setting may cause a large number of threads to be created and have a negative impact on system storage and performance.

MaxKeepAliveConnections

The MaxKeepAliveConnections parameter describes the maximum number of concurrent connections to the Web container that are allowed to be kept alive, that is to process multiple requests from the same client. The Web server plug-in keeps connections open to the

application server as long as it can. This type of connection is called a *persistent connection*. The persistent connection improves the response time.

However, if the value of this property is too small, performance is negatively impacted because the plug-in has to open a new connection for each request instead of sending multiple requests from the same client through one connection. The application server may not accept a new connection under a heavy load if there are too many sockets in TIME_WAIT state.

If all client requests are going through the Web server plug-in and there are many TIME_WAIT state sockets for your internal HTTP port, the application server is closing connections prematurely, which decreases performance. The application server closes the connection from the plug-in, or from any client, for any of the following reasons:

- ▶ The client request was an HTTP 1.0 request when the Web server plug-in sent HTTP 1.1 requests.
- ▶ The maximum number of concurrent keep-alives was reached. A keep-alive must be obtained only once for the life-time of a connection, that is after the first request is completed, but before the second request can be read.
- ▶ The maximum number of requests for a connection was reached, preventing denial of service (DoS) attacks in which a client tries to hold on to a keep-alive connection forever.
- ▶ A timeout occurred while waiting to read the next request or to read the remainder of the current request.

Setting MaxKeepAliveConnections

To change or set this WebSphere Application Server value, follow these steps:

1. Start the administrative console.
2. In the topology tree, expand **Servers** and click **Application Servers**.
3. Click the name of the application server that you want to configure.
4. Click **Web Container**.
5. On the Web Container page, click **HTTP transports**.
6. Select your **Port**.
7. On your Port page, scroll down and click **Custom Properties**.
8. To add a custom property, follow these steps:
 - a. Click **New**.
 - b. Specify a name and value for the property. For example, as shown in Figure 8-9, we set Name to MaxKeepAliveConnections and Value to 30. You can also specify a description.
 - c. After you specify the custom property or properties, click **OK**.
9. Save the configuration.
10. Restart the application server.

[Application Servers](#) > [anakin](#) > [Web Container](#) > [HTTP Transport](#) > [11109](#) > [Custom Properties](#) >

New

Specifies arbitrary name/value pairs of data, where the name is a property key and the value is a string value which can be used to set internal system configuration properties. [i](#)

Configuration

| General Properties | | |
|--------------------|----------------------------|---|
| Name | * MaxKeepAliveConnections | i Specifies the name (or key) for the property. |
| Value | * 30 | i Specifies the value paired with the specified name. |
| Description | Max Keep Alive Connections | i Provides information about the name-value pair. |

Apply OK Reset Cancel

Figure 8-9 HTTP Transport Custom Properties setting

Default value

A default value is not set for MaxKeepAliveConnections.

Recommended value

The value should be at most 90% of the maximum number of threads in the Web container thread pool. If it is 100% of the maximum number of threads in the Web container thread pool, all the threads could be consumed by keep-alive connections, leaving no threads available to process new connections. One of several possible problems with allocating all the threads to the persistent connections is a DoS attack.

MaxKeepAliveRequests

The MaxKeepAliveRequest value controls the maximum number of requests allowed on a single keep-alive connection. This parameter can help prevent DoS attacks when a client tries to hold on to a keep-alive connection. The Web server plug-in keeps connections open to the application server as long as it can, providing optimum performance.

Setting MaxKeepAliveRequests

To change or set MaxKeepAliveRequests, follow these steps:

1. Start the administrative console.
2. In the topology tree, expand **Servers** and click **Application Servers**.
3. Click the name of the application server that you want to configure.
4. Click **Web Container**.
5. On the Web Container page, click **HTTP transports**.
6. Select your **Port**.
7. On your Port page, scroll down and click **Custom Properties**.
8. To add a custom property, follow these steps:
 - a. Click **New**.
 - b. Specify a name and value for the property. In this example, we specify, MaxKeepAliveRequests for Name and 100 for Value as shown in Figure 8-10.

[Application Servers](#) > [anakin](#) > [Web Container](#) > [HTTP Transport](#) > [11109](#) > [Custom Properties](#) >

New

Specifies arbitrary name/value pairs of data, where the name is a property key and the value is a string value which can be used to set internal system configuration properties. [i](#)

Configuration

General Properties

| | | |
|-------------|-------------------------|---|
| Name | * MaxKeepAliveRequests | i Specifies the name (or key) for the property. |
| Value | * 100 | i Specifies the value paired with the specified name. |
| Description | Max Keep Alive Requests | i Provides information about the name-value pair. |

Apply OK Reset Cancel

Figure 8-10 HTTP Transport Custom Properties setting

- c. You can also specify a description as shown in Figure 8-11.

[Application Servers](#) > [anakin](#) > [Web Container](#) > [HTTP Transport](#) > [11109](#) >

Custom Properties

Specifies arbitrary name/value pairs of data, where the name is a property key and the value is a string value which can be used to set internal system configuration properties. [i](#)

Total: 2

[Filter](#)

[Preferences](#)

New Delete

| <input type="checkbox"/> | Name ▾ | Value ▾ | Description ▾ |
|--------------------------|---|---------|----------------------------|
| <input type="checkbox"/> | MaxKeepAliveConnections | 30 | Max Keep Alive Connections |
| <input type="checkbox"/> | MaxKeepAliveRequests | 100 | Max Keep Alive Requests |

Figure 8-11 HTTP Transport Custom Properties

- d. After you specify the custom property or properties, click **OK**.
9. Save the configuration.
10. Restart the application server.

Default value

A default value is not set for MaxKeepAliveRequests.

Recommended value

A good starting value is 100. If the application server requests are received from the plug-in only, increase this parameter's value.

8.2.2 Session management

WebSphere Application Server session support has features for tuning session performance and operating characteristics, particularly when sessions are configured in a distributed environment. These options support the administrator's flexibility in determining the performance and failover characteristics for his environment.

Maximum in-memory session count

You set the Maximum in-memory session count value to a number that is 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. Be sure that Allow Overflow is enabled, which is enabled by default. You can use the Tivoli Performance Viewer to monitor this setting. See Figure 8-12.

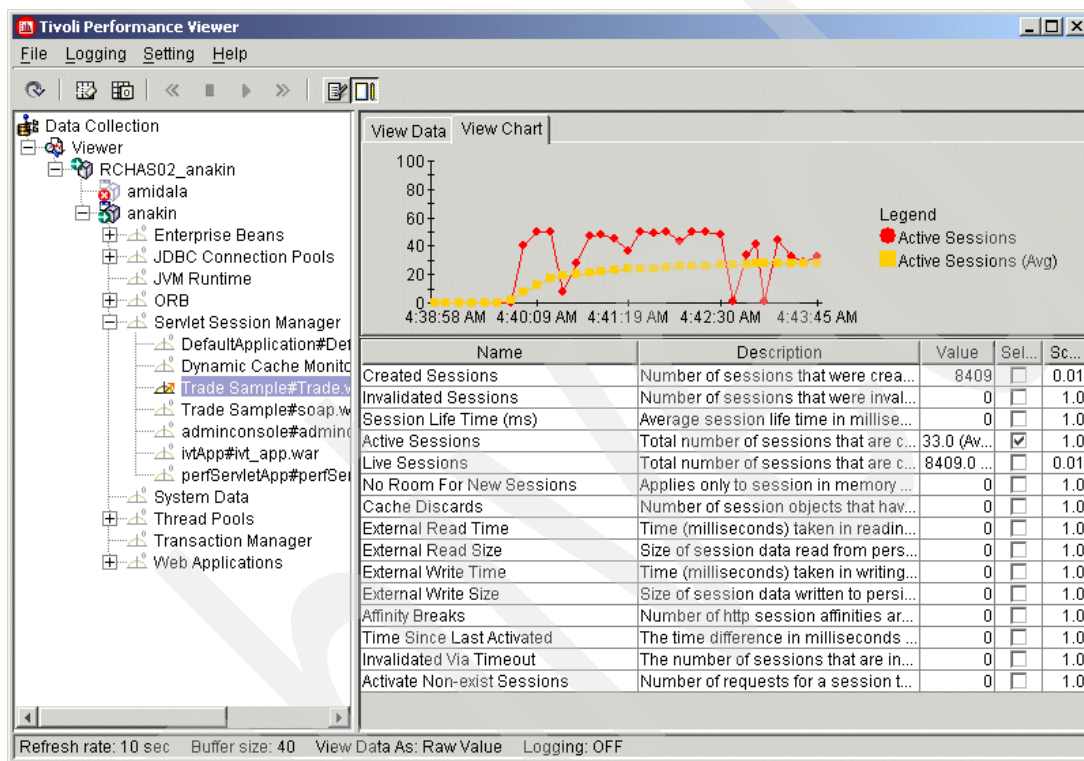


Figure 8-12 Session management: Active sessions


Setting Maximum in-memory session count

To change or set this WebSphere Application Server value, use these steps.

1. Start the administrative console.
2. In the topology tree, expand **Servers** and click **Application Servers**.
3. Click the name of the application server that you want to configure.
4. Click **Web Container**.
5. On the Web Container page, click **Session Management**.
6. In the Maximum in-memory session count field, specify a value and select **Allow overflow**. See Figure 8-13.
7. Click **Apply** or **OK**.
8. Save the configuration.
9. Restart the application server.

[Application Servers](#) > [anakin](#) > [Web Container](#) >

Session Management

Session manager configuration properties allow you to control the behavior of HTTP session support 







| Configuration | | |
|----------------------------------|---|--|
| General Properties | | |
| Session tracking mechanism: | <input type="checkbox"/> Enable SSL ID tracking <input checked="" type="checkbox"/> Enable Cookies <input type="checkbox"/> Enable URL Rewriting <input type="checkbox"/> Enable protocol switch rewriting |  Specify a mechanism for HTTP session management. |
| Maximum in-memory session count: | <input type="text" value="1000"/> sessions |  Specifies the maximum number of sessions to maintain in memory. |
| Overflow: | <input checked="" type="checkbox"/> Allow overflow |  Whether to allow the number of sessions in memory to exceed the value specified by Max In Memory Session Count property. This is valid only in non-persistent sessions mode. |
| Session timeout: | <input type="radio"/> No timeout <input checked="" type="radio"/> Set timeout <input type="text" value="30"/> minutes |  Specifies how long a session is allowed to go unused before it will be considered valid no longer. Specify either "Set timeout" or "No timeout." If you select to set the timeout, the value must be at least two minutes, specified in minutes. |
| Security integration | <input type="checkbox"/> Enable |  When security integration is enabled, the Session Manager will associate the identity of users with their HTTP sessions. |
| Serialize session access: | <input type="checkbox"/> Allow serial access Maximum wait time : <input type="text" value="5"/> seconds <input checked="" type="checkbox"/> Allow access on timeout |  Serialize session access indicates whether to disallow concurrent session access in a given server (JVM). |

Figure 8-13 Session management settings

Default value

The default value for the Maximum in-memory session count field is 1000. Be sure this setting is big enough and check it with the Tivoli Performance Viewer. Table 8-2 shows the results for 100 clients.

Table 8-2 Maximum in-memory session count test with 100 clients

| | First test | Second test | Third test |
|------------------------------|------------|-------------|------------|
| Maximum in-memory session | 1000 | 150 | 2000 |
| HTTP Requests | 15736 | 15841 | 15915 |
| Average Request Latency (ms) | 1.982 | 1.997 | 1.869 |

Serialize session access

This value specifies that concurrent session access in a given server is not allowed. In a distributed environment, the session manager requires an affinity mechanism so that requests for a particular session are always sent to the same application server each time. This satisfies the Servlet 2.3 requirement that a session can be active only in one JVM at a time. It also provides for better performance, because the sessions are cached in memory. We did a test by allowing this setting and achieved a result with a better response time. See Table 8-3.

Table 8-3 Serialize session access test with 100 clients

| | First test | Second test |
|------------------------------|------------|-------------|
| Allow serial access | No | Yes |
| HTTP Requests | 15851 | 15839 |
| Average Request Latency (ms) | 1.557 | 1.344 |

Default value

The default value for Serialize session access is no.

Schedule sessions cleanup

If possible, invalidate finished sessions promptly in your application. You can do this by using the `javax.servlet.http.HttpSession.invalidate()` method. Also set Invalidation Time-out to the smallest value that is acceptable for your application. To view or set this property, see Figure 8-13.

Distributed sessions

Do not enable persistent sessions unless required by the application. If persistent sessions are required, enable multi-row sessions.

8.2.3 Dynamic cache service

WebSphere Application Server consolidates several caching activities, including servlets, JavaServer Pages (JSP), Web services, and WebSphere Application Server commands into one service called the *dynamic cache service*.

Caching the output of servlets, commands, and JSP files can improve application performance. WebSphere Application Server consolidates several caching activities, including servlets, Web services, and WebSphere Application Server commands into one service called the dynamic cache service. Caching activities share many configuration parameters, which are set in the dynamic cache service.

The dynamic cache service works within an application server, JVM and intercepts calls to cacheable objects. For example, it intercepts calls through a servlet's `service()` method or a command's `execute()` method. It then stores the output of the object in (or serves the content of the object from) the dynamic cache.

You can use the dynamic cache service to improve the performance of servlet and JSP files by serving requests from an in-memory cache. Cache entries contain servlet output, results of servlet execution, and metadata.

For example, a Web site that sells toys can use dynamic caching to cache the servlet that looks up information about a popular toy. It is unlikely that the information displayed about the toy changes often, which makes data that is 10 or more minutes old acceptable. When WebSphere Application Server receives the request, it immediately returns the cached output instead of executing the servlet or JSP code. This results in a faster response for the user, less servlet processing, and less overhead consumed by queries to the database.

It is important that you use dynamic caching only when appropriate, because caching servlets that are rarely used may consume more resources than it saves.

The following sections explain how to enable and configure the dynamic cache service. They also explain how to use the advanced features, such as to configure external caches and build user-defined drop-in components to customize the cache operation:

Enabling dynamic caching

Use the dynamic cache service to improve the performance of cacheable objects. By default, the dynamic cache service is enabled.

Perform the following steps to enable dynamic caching:

1. Start the WebSphere Application Server administrative console.
2. In the topology tree, expand **Servers** and click **Application Servers**.
3. Click the server you want to enable dynamic cache.
4. Click **Dynamic Cache Service**.
5. In the Startup state field, select **Enable service at server startup**. See Figure 8-14.

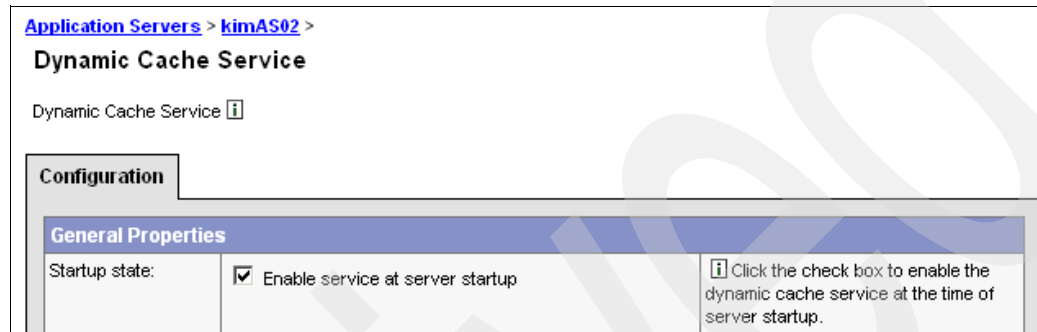


Figure 8-14 Enable service at server startup

6. Click **Apply** or **OK**.
7. Save the administrative configuration.
8. Restart the application server.

When dynamic cache is enabled, you see the following message in SystemOut.log:

```
CacheServiceI I DYNA0048I: WebSphere Dynamic Cache initialized successfully
```

And on every application that has dynamic cache defined, you see:

```
ConfigManager I DYNA0047I: Successfully loaded cache configuration file  
/QIBM/UserData/WebAS51/Base/anakin/installedApps/AS02B_anakin/trade3.ear/trade3Web.war/WEB-INF/cachespec.xml
```

Configuring dynamic caching

The following sections explain how to configure caching for various WebSphere Application Server components.

Enabling servlet caching


After a servlet is invoked, the output to be cached is generated, and a cache entry is created. The cache entry contains the output of the servlet and the side effects of the invocation. For example, side effects include calls to other servlets or JSP files, as well as metadata about the entry, including timeout and entry priority information.

Unique entries are distinguished by an ID string generated from the `HttpServletRequest` object for each invocation of the servlet. You can base servlet caching on request parameters and attributes, the URI used to invoke the servlet, session information, or other options, including cookies.

Note: To enable servlet caching, the dynamic cache service must also be enabled.

To enable servlet and JSP caching, follow these steps:

1. Start the WebSphere Application Server administrative console.
2. In the topology tree, expand **Servers** and click **Application Servers**.
3. Click the server you want to work with.
4. Click **Web Container**.
5. Under the Configuration tab, select the **Enable servlet caching** check box. See Figure 8-15.



The screenshot shows the 'Web Container' configuration page for server 'kimAS02'. The 'Configuration' tab is active. Under the 'General Properties' section, the 'Servlet caching' checkbox is checked, and the 'Enable servlet caching' button is highlighted. The 'Default virtual host' is set to 'default_host'. The 'Apply' button is also visible.

Figure 8-15 Enable servlet caching

6. Click **Apply** or **OK**.
7. Save the administrative configuration.
8. Restart the application server.

Setting Cache size

The Cache size parameter represents the number of the cached objects. Specify a positive integer as the value for the maximum number of entries the cache holds. Enter the cache size value in this field between the range of 100 and 200000.

Perform the following steps to change the cache size:

1. Start the WebSphere Application Server administrative console.
2. In the topology tree, expand **Servers** and click **Application Servers**.
3. Click the server you want to enable dynamic cache.
4. Click **Dynamic Cache Service**.
5. In the Cache Size field, specify a value. See Figure 8-16.
6. Click **Apply** or **OK**.
7. Save the administrative configuration.
8. Restart the application server.

[Application Servers](#) > [anakin](#) >

Dynamic Cache Service

Dynamic Cache Service ⓘ

Configuration

| General Properties | | |
|--------------------|--|--|
| Startup state: | <input checked="" type="checkbox"/> Enable service at server startup | ⓘ Click the check box to enable the dynamic cache service at the time of server startup. |
| Cache size | * 2000 entries | ⓘ A positive integer defining the maximum number of entries the cache will hold. |
| Default Priority | * 1 | ⓘ The default priority for cache entries, determining how long an entry will stay in a full cache. |

Figure 8-16 Cache size setting

Default value

The default value for Cache size is 2000.

Recommended value

When the number of cache entries reaches the configured limit for WebSphere Application Server, eviction of cache entries occurs. This allows new entries to enter the cache service. The dynamic cache service includes an alternative feature named disk offload, which copies the evicted cache entries to disk for potential future access. See more at “Advantages of dynamic cache with Trade 3.1” on page 256.

Configuring cacheable objects with the cachespec.xml file

Define cacheable objects inside the cachespec.xml file. You can find this file in the Web module WEB-INF directory or the enterprise bean META-INF directory.

We recommend that you place the cache configuration file with the deployment module. This allows for better organization of caching schemas and easier redeployment, if required. However, you can place a global cachespec.xml in the application server properties directory. The root element of the cachespec.xml file is <cache>, which contains <cache-entry> elements.

Within a <cache-entry>...</cache-entry> element are parameters that enable the dynamic cache using the cachespec.xml file:

1. Develop a cachespec.xml file.
 - a. Create a caching configuration file.
 - b. In the /QIBM/ProdData/WebAS51/Base/properties directory, locate the **cachespec.sample.xml** file.
 - c. Copy the cachespec.sample.xml file to cachespec.xml in Web module WEB-INF or enterprise bean META-INF directory.
2. Define the cache-entry elements necessary to identify the servlet. See Example 8-1.

Example 8-1 Cache entry example

```
<class>servlet</class>
<name>/displayQuote.jsp</name>
<property name="EdgeCacheable">true</property>
```

3. Develop cache-ID rules.

To cache an object, WebSphere Application Server must know how to generate unique IDs for different invocations of that object. The `<cache-id>` element performs that task. Each cache entry can have multiple cache-ID rules that execute until a rule returns a non-empty cache-ID or no more rules remain to execute. If none of the cache-ID generation rules produce a valid cache ID, then the object is not cached. Develop these IDs in one of two ways:

- Use the `<component>` element, which is defined in the cache policy of a cache entry (recommended).
- Write custom Java code to build the ID from input variables and system state.

To configure the cache entry using the `<component>` element, specify your ID in the Extensible Markup Language (XML) file (see Example 8-2). `<priority>` and `<timeout>` elements define how long a cached object stays in cache. That is, the higher the value is, the longer it stays in the cache.

Example 8-2 Cache ID example

```
<cache-id>
  <component id="symbol" type="parameter">
    <required>true</required>
  </component>
  <priority>3</priority>
  <timeout>180</timeout>
</cache-id>
```

4. Specify dependency ID rules.

Use dependency ID elements to specify additional cache group identifiers that associate multiple cache entries to the same group identifier.

The dependency ID is generated by concatenating the dependency ID base string with the values returned by its component elements. If a required component returns a null value, the entire dependency ID does not generate and is not used. You can validate the dependency IDs explicitly through the WebSphere Application Server Dynamic Cache application programming interface (API), or by using another cache-entry `<invalidation>` element. Multiple dependency ID rules can exist per cache-entry. All dependency ID rules separately execute. See Example 8-3.

Example 8-3 Dependency ID example

```
<dependency-id>Quote_Symbol
  <component id="symbol" type="parameter">
    <required>true</required>
  </component>
</dependency-id>
```

5. Invalidate other cache entries as a side effect of this object execution, if relevant.

Define invalidation rules the same way as dependency IDs. The IDs that generate by invalidation rules are used to invalidate cache entries that have those IDs.

The invalidation ID is generated by concatenating the invalidation ID base string with the values returned by its component element. If a required component returns a null value, then the entire invalidation ID is not generated and no invalidation occurs. Multiple invalidation rules can exist per cache-entry. All invalidation rules separately execute.

6. Verify the cacheable page.

Typically, you declare several `<cache-entry>...</cache-entry>` elements inside a `cachespec.xml` file.

The dynamic cache responds to changes in this file at run-time. When new versions of the `cachespec.xml` file are detected, the old policies are replaced. Objects cached through the old policy file are not automatically invalidated from the cache; they are either reused with the new policy or eliminated from the cache through its replacement algorithm.

For each of the three IDs (cache, dependency, invalidation) that generate by cache entries, a `<cache-entry>` can contain multiple elements. The dynamic cache executes the `<cache-id>` rules in order. The first rule that successfully generates an ID is used to cache that output. If the object should be cached, each of the `<dependency-id>` elements are executed to build a set of dependency IDs for that cache entry. Finally, each of the `<invalidation>` elements are executed, building a list of IDs that the dynamic cache invalidates, regardless of whether this object is cached. For more information, see the WebSphere Application Server Information Center Web site for Version 5.1:

http://publib.boulder.ibm.com/infocenter/ws51help/index.jsp?topic=/com.ibm.wasee.doc/info/ee/ae/rdyn_cachespec.html

Dynamic Cache Policy Editor

The Dynamic Cache Policy Editor makes cache policies easier to create and maintain by providing an interface that plugs into WebSphere Studio Application Developer or Application Server Toolkit. The editor validates cache policies against its XML schema, provides assistance in completing content, and restricts changes that are invalid during run time. The plug-in also includes a simple tool that can analyze servlets and JSPs and then generate the cache policies and add them to the cache policy file. You find more information at [alphaWorks®](http://www.alphaworks.ibm.com/tech/cachepolicyeditor):

<http://www.alphaworks.ibm.com/tech/cachepolicyeditor>

As a sample of a defined `cachespec.xml`, the Trade 3.1 application has the `cachespec.xml` file (see Figure 8-17).

Consider these notes on dynamic caching setup:

- ▶ The number of cache entries is specified in the `dynacache.xml` file.
- ▶ The cached items are stored in the JVM heap of the application server.
- ▶ Because there is little overhead for managing cached items, add the total cache output HTML size to the JVM (5000 items at 10K in size = increase maximum heap by 50 MB).
- ▶ A priority value of 1 insures a cached value remains in the cache at least once as the Least Recently Used algorithm determines which values to remove from the cache.
- ▶ At a high volume Web site where the average output of the cached entries is difficult to determine or certain group of servlet and JSP calls are called very frequently, the priority value should be increased.
- ▶ Priority values should not be set too high (greater than 3) since too much time is spent calculating Least Recently Used values for expiring cache entries.
- ▶ Caches can also be available to other applications, such as WebSphere Edge Server and Adaptive Fast Path Architecture (AFPA), which allow for caching requests before they even reach WebSphere Application Server. This greatly improves performance as the request reaches the HTTP server and the load balancing server can be responded to without reaching the application server.
- ▶ As with caching on the application server, only complete pages are cached based on unique URLs and parameters.

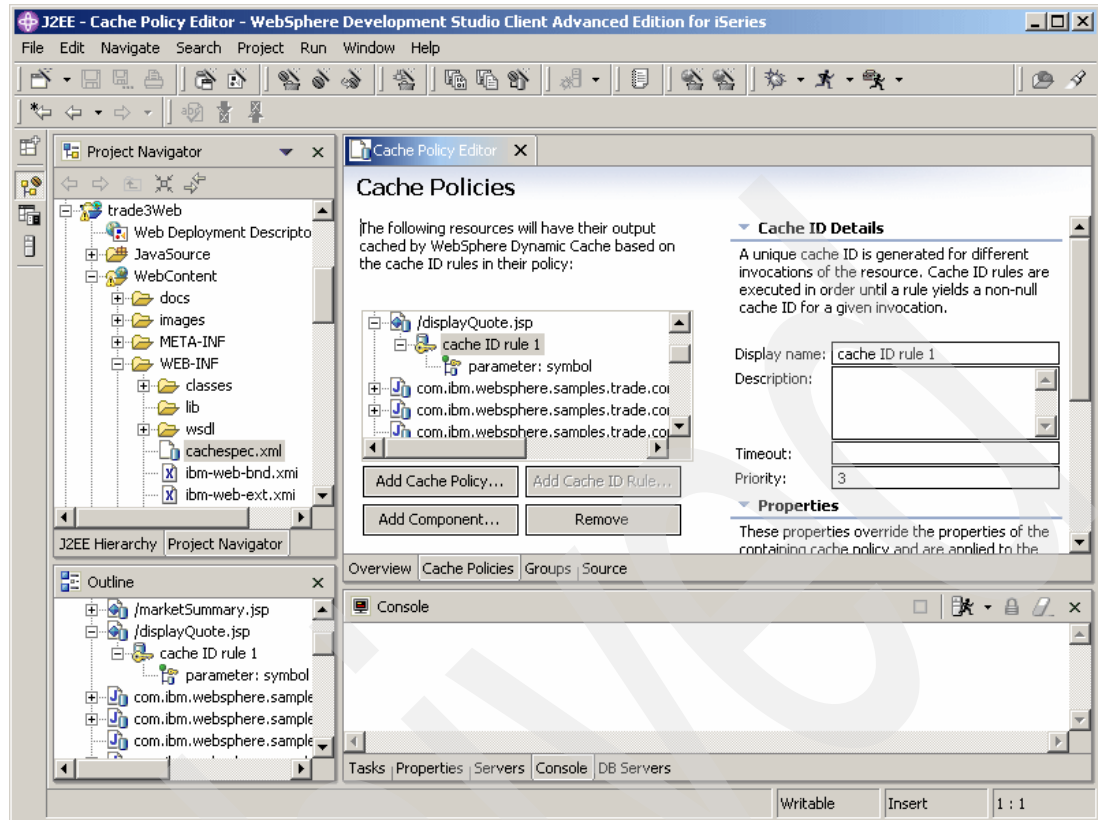


Figure 8-17 Dynamic Cache Policy Editor for WebSphere Application Server

Cache monitor

Cache monitor is an installable Web application that provides a real-time view of the current state of dynamic cache. You use it to help verify that dynamic cache is operating as expected. The only way to manipulate the data in the cache is by using the cache monitor. It provides a graphical user interface (GUI) to manually change data.

Cache monitor provides a way to:

- ▶ Verify the configuration of dynamic cache

The WebSphere Application Server administrative console provides ways to enable the dynamic cache service and configure properties, such as maximum size of the cache and disk offload location, as well as advanced features such as controlling external caches. Cache monitor offers a way for dynamic cache users to verify the configuration of the dynamic cache by providing a convenient view of the configured features and properties in the cache monitor.

- ▶ Verify the cache policies

To cache an object, WebSphere Application Server must know how to generate unique IDs for different invocations of that object. This is performed by providing rules for each cacheable object in the cachespec.xml file. This file is found inside the Web module WEB-INF or enterprise bean META-INF directory. Each cacheable object can have multiple cache ID rules that execute in sequence until either a rule returns a cache ID or no more rules remain to execute. If none of the cache ID generation rules produce a valid cache ID, then the object is not cached.

Since there can be multiple cachespec.xml files with multiple cache ID rules, cache monitor provides a convenient way to verify the policies of each object. It offers a view of

all the cache policies currently loaded in dynamic cache. This view is also convenient to verify that the cachespec.xml file was read by the dynamic cache without errors. In Figure 8-18, under Cache Policies, see our example for Trade 3.1.

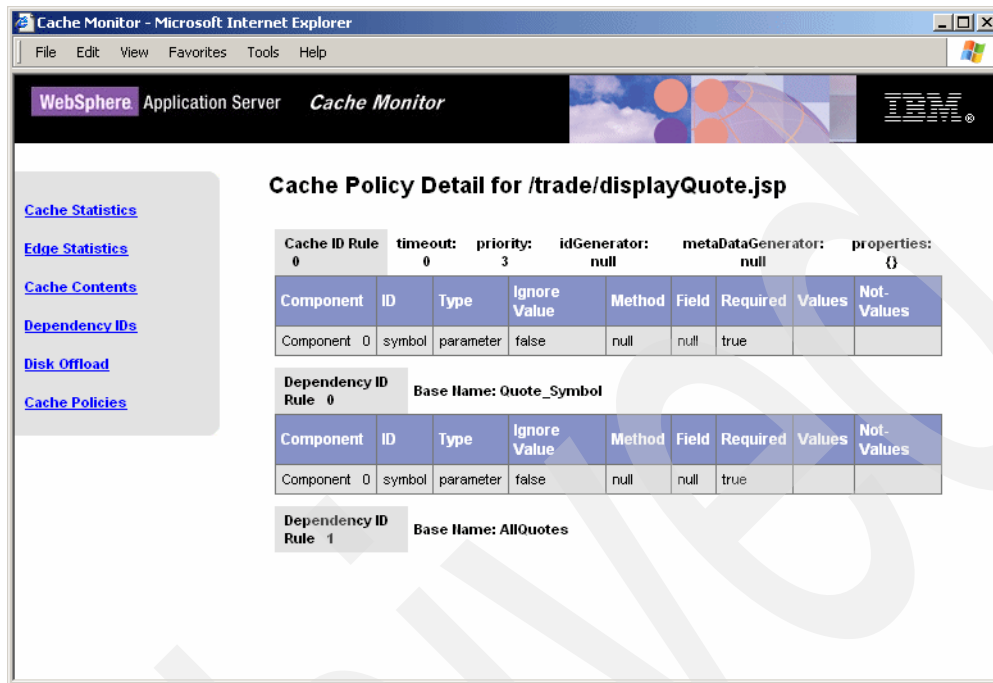


Figure 8-18 Current cache policies for displayQuote

► Monitor cache statistics

Cache monitor provides a view of the essential cache data, such as number of cache hits, cache misses, and number of entries in cache. This helps to tune the cache configuration optimally to get the best performance improvement out of dynamic cache. For example, if the number of used entries is often high, and entries are being removed and recreated, consider increasing the maximum size of the cache or enabling disk offload. See Figure 8-19.

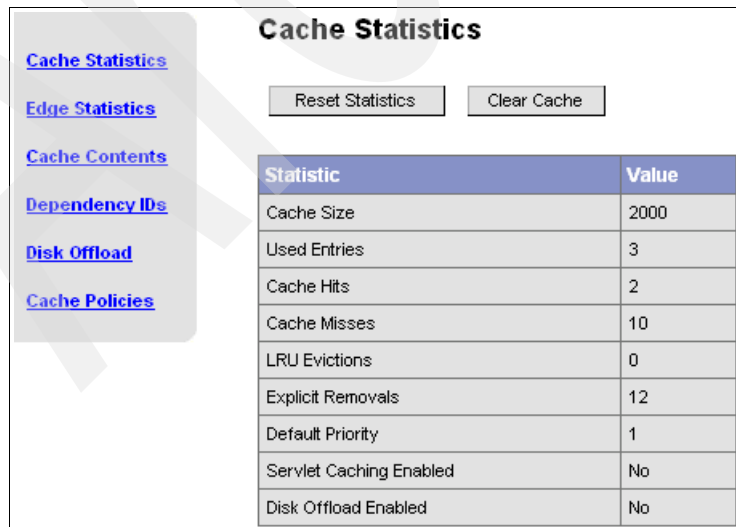


Figure 8-19 Monitor cache statistics

► Monitor the data flowing through the cache

When a cacheable object is invoked, dynamic cache creates a cache entry for it that contains the output of the execution and metadata, such as time to live, sharing policy, and so on. Entries are distinguished by a unique ID string that is based on the rules specified in the `cachespec.xml` file for this object's name. Objects with the same name may generate multiple cache IDs for different invocations, based on request parameters and attributes for each invocation.

Cache monitor provides a view of all the cache entries currently in cache, based on the unique ID. It also provides a view of the group of cache entries that share a common name (also known as template). Cache entries can also be grouped together by a dependency ID, which is used to invalidate the entire group of entries dependent on a common entity. Therefore, cache monitor also provides a view of the group of cache entries that share a common dependency ID.

For each entry, cache monitor also displays metadata, such as time to live, priority and sharing-policy, and provides a view of the output that has been cached. This helps the customer to verify which pages are cached, that the pages have been cached with the right attributes such as time to live, priority, and so on, and that the pages have the right content.

► Monitor the data in the edge cache

Dynamic cache provides support to recognize the presence of an Edge Side Include (ESI) processor and to generate ESI include tags and appropriate cache policies for edge cacheable fragments. The ESI processor has the ability to cache whole pages, as well as fragments, providing a higher cache hit ratio. There can be multiple ESI processors running on multiple hosts configured for caching.

Cache monitor provides a list of all ESI processes and their hosts that are enabled for caching. It also provides a way to select a host or a processor, and view its edge cache statistics as well as current cache entries. For more information, see edge cache statistics in the Information Center at:

<http://publib.boulder.ibm.com/was400/51/english/info/rzaiz/51/program/servdyncomonesi.htm>

► View the data offloaded to the disk

By default, when the number of cache entries reaches the configured limit for a given server, eviction of cache entries occurs. This allows new entries to enter the cache service. The dynamic cache includes the *disk offload* feature that copies the evicted cache entries to disk for future access. Cache monitor offers a view of the content offloaded to disk that corresponds to the view of contents cached in memory.

► Manage the data in the cache

In addition to displaying cache content, cache monitor provides basic operations on the data in the cache:

- Removing an entry from the cache
- Removing all entries for a certain dependency ID
- Removing all entries for a certain name (template)
- Moving an entry to the front of the least recently used queue to avoid eviction
- Moving an entry from the disk to the cache
- Clearing the entire contents of the cache
- Clearing the contents of the disk cache

These functions are useful for dynamic cache customers. They provide a way to manually change the state of the cache without restarting the server.

Advantages of dynamic cache with Trade 3.1

The Trade 3.1 application uses dynamic cache. Table 8-4 shows the results of our test. Be careful when you interpret these results. Not all applications can benefit from dynamic cache in the same way.

Table 8-4 Dynamic cache test with 100 clients

| Maximum in memory cache size | 2000 | 1000 | 500 | 0 |
|------------------------------|-------|-------|-------|-------|
| In Memory Cache Size | 1001 | 1000 | 5000 | 0 |
| Servlet Cache enabled | Yes | Yes | Yes | No |
| HTTP Requests | 70210 | 72009 | 44690 | 14051 |
| Average Request Latency | 0.323 | 0.349 | 0.494 | 1.075 |

To read the In Memory Cache Size value, see Figure 8-20.

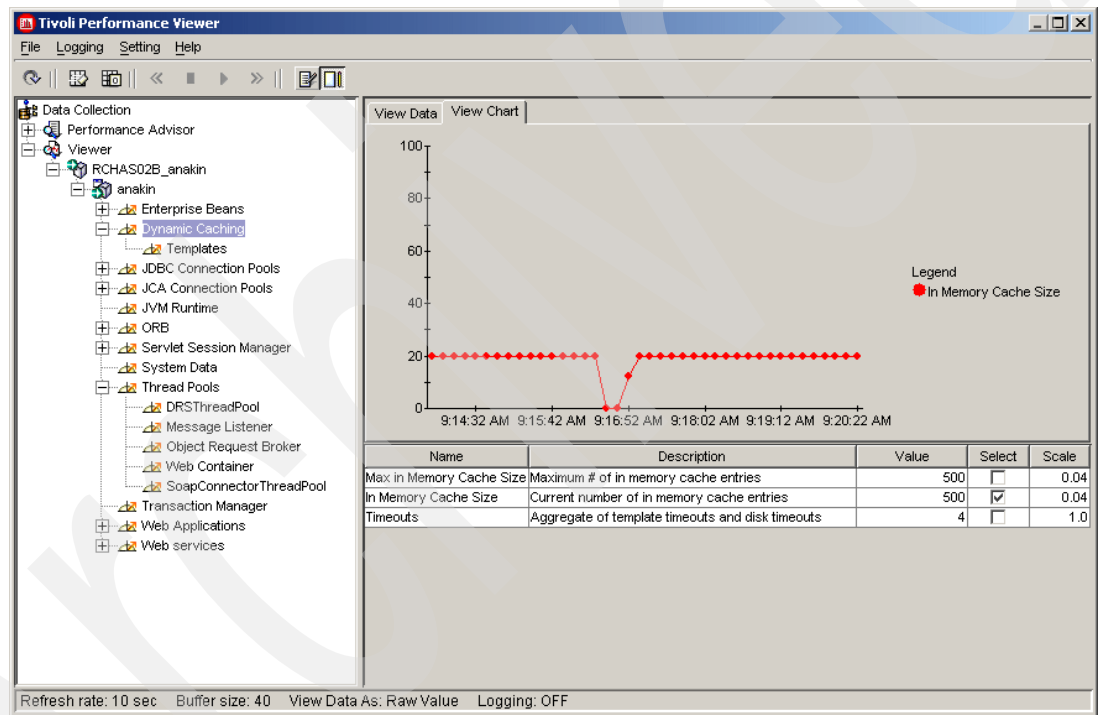


Figure 8-20 Dynamic cache: In Memory Cache Size at the TPV

An other way to keep track of the dynamic cache is the Cache Monitor application. During the test, we received the result shown in Figure 8-21.

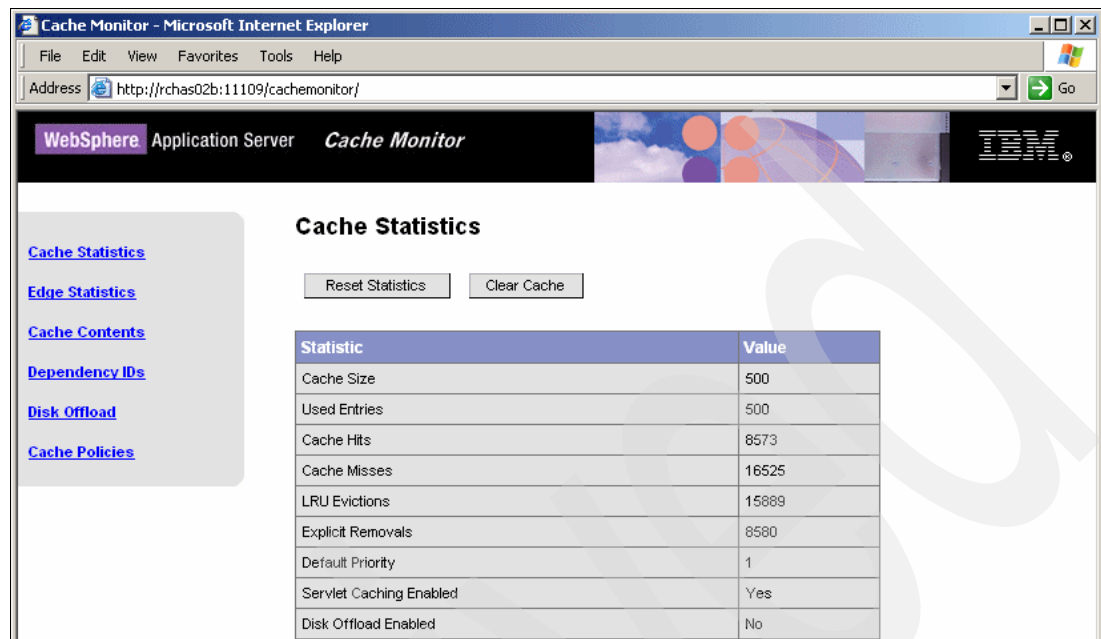


Figure 8-21 Cache Monitor application result during the tests

Configuring external caching

The dynamic cache service has the ability to control caches outside of the application server, such as the IBM Edge Server cache and IBM HTTP Server cache (Fast Response Cache Accelerator (FRCA)). When you define external cache groups, the dynamic cache service matches externally cacheable cache entries with those groups and pushes cache entries and invalidations out to those groups. This allows WebSphere Application Server to manage dynamic content beyond the application server. The content can be served from the external cache instead of the application server, which can improve performance.

For more information, see the following Web site from the WebSphere Application Server for iSeries Information Center Version 5.1:

<http://publib.boulder.ibm.com/was400/51/english/info/rzaiz/51/program/servdynext.htm>

Cache replication

Cache replication allows data to be generated one time and copied or replicated to other servers in the cluster. Caching in a cluster has additional concerns. In particular, the same data may be required, and therefore, generated in multiple places. Also, the access that resources need to generate the cached data can be restricted, preventing access to the data.

For more information, see the following Web site from the WebSphere Application Server for iSeries Information Center Version 5.1:

<http://publib.boulder.ibm.com/was400/51/english/info/rzaiz/51/program/servdynccacherep.htm>

Troubleshooting the dynamic cache service

Complete these steps to resolve problems that you think are related to the dynamic cache service:

1. Review the JVM logs for your application server. Messages that have the preface DYNA result from dynamic cache service operations.
 - a. View the JVM logs for your application server. Each server has its own JVM log file. For example, if your server is named Member_1, the JVM log is located in the subdirectory `<install_root>/logs/Member_1`.

To use the administration console to review the JVM logs, click **Troubleshooting** → **Logs and Trace** → **server_name** → **JVM Logs** → **Runtime** → **View**.
 - b. Find any messages prefaced with DYNA in the JVM logs, and write down the message IDs.
 - c. Find the message for each message ID.
 - d. Read the message Explanation and User Action statements.
 - e. Try the solutions stated under User Action in the DYNA messages.
2. Use the cache monitor to determine whether the dynamic cache service is functioning as expected. The cache monitor is an installable Web application that displays simple cache statistics, cache entries, and cache policy information.

You can find the descriptions of the WebSphere Dynacache messages on the Web at:

<http://publib.boulder.ibm.com/was400/51/english/info/rzaiz/51/trb/help/dyna.htm>

8.2.4 Data sources

Applications use data sources to access databases. The following datasource settings can affect performance:

- ▶ Connection pooling
- ▶ Statement cache size

Connection pooling

When accessing any database, the initial database connection is an expensive operation. WebSphere Application Server supports JDBC 2.0 Standard Extension APIs to provide support for connection pooling and connection reuse. The connection pool is used for efficient use of the database connections.

Tivoli Performance Viewer can help to find the optimal size for the connection pool. Use a standard workload that represents a typical number of incoming client requests. Use a fixed number of iterations and a standard set of configuration settings. Watch the Pool Size, Percent Used and Concurrent Waiters counters of the data source entry under the JDBC Connection Pools module.

When the application reaches a stable state, look for connections that are being closed instead of returned to the pool. The optimal value for the pool size is that which reduces the values for these monitored counters. If Percent Used is consistently low, consider decreasing the number of connections in the pool. See Figure 8-22 as a sample.

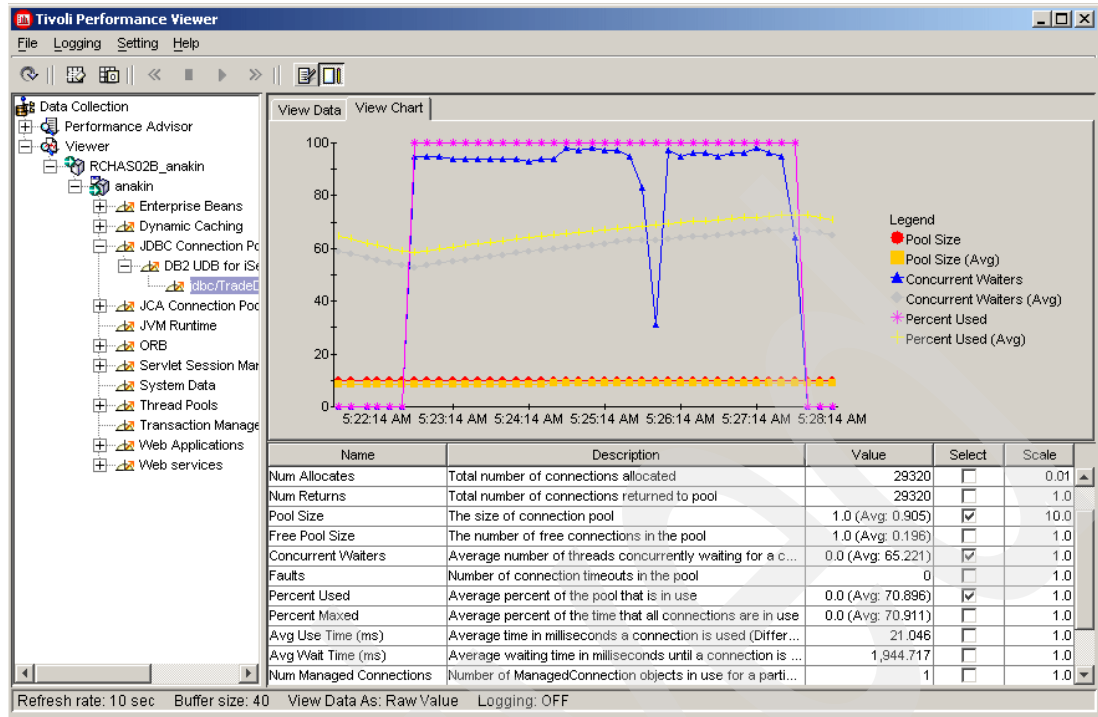


Figure 8-22 Connection pool size

Better performance is generally achieved if the value for the connection pool size is set lower than the value for the Max Connections in the Web container. Lower settings for the connection pool size (10 to 30 connections) typically perform better than higher (more than 100) settings.

Keep in mind that a data source connection takes 2 MB 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 the maximum number of connections. Remember that not all of the requests that reach the servlet engine generate the same database workload. Some requests may not perform any database input/output (I/O) at all.

Each entity bean transaction requires an additional connection to the database specifically to handle the transaction. Take this into account when calculating the number of data source connections.

Maximum connection pool

The Maximum connection pool value specifies the maximum number of managed connections for a pool.

Setting Maximum connection pool

Follow these steps:

1. Start the administrative console.
2. In the topology tree, expand **Resources** and click **JDBC Providers**.
3. Click the name of the provider for the data source that you want to configure.
4. Click **Data Sources**.
5. Click the name of the data source that you want to configure.
6. Click **Connection Pool**.
7. In the Max Connections field, specify a value. See Figure 8-23.

[JDBC Providers](#) > [DB2 UDB for iSeries \(Native XA - V5R2 and later\)](#) > [Data Sources](#) > [TradeDataSource](#) >

Connection Pools

Connection pool properties that can be modified to change the behavior of the J2C connection pool manager. Default values are provided for non-production use. Reviewing and possible modification of these configuration values is recommended. [i](#)

Configuration

| General Properties | | |
|--------------------|---|---|
| Scope | cells:RCHAS02B_anakin:nodes:RCHAS02B_anakin | i The scope of the configured resource. This value indicates the configuration location for the configuration file. |
| Connection Timeout | 1800 seconds | i Interval, in seconds, after which a connection request times out and a <code>ConnectionWaitTimeoutException</code> is thrown. |
| Max Connections | 100 connections | i The maximum number of <code>ManagedConnections</code> that can be created in this pool. |
| Min Connections | 100 connections | i The minimum number of <code>ManagedConnections</code> that should be maintained. |
| Reap Time | 18000 seconds | i Interval, in seconds, between runs of the pool maintenance thread. |
| Unused Timeout | 1800 seconds | i Interval, in seconds, after which an unused connection is discarded by the pool maintenance thread. |
| Aged Timeout | 0 seconds | i Interval, in seconds, after which an unused, aged connection is discarded (regardless of recent usage activity) by the pool maintenance thread. |
| Purge Policy | FailingConnectionOnly | i Specifies how to purge connections when a "stale connection" or "fatal connection error" is detected. |

Apply
OK
Reset
Cancel

Figure 8-23 Connection Pools settings

8. Click **Apply** or **OK**.
9. Save the configuration.
10. Restart the application server.

Default value

The default value for Maximum connections is 10.

Minimum connection pool

The Minimum connection pool value specifies the minimum number of managed connections for a pool. The data source tries to keep at least this number of open connections to the database.

Setting Minimum connection pool

To set this value, follow these steps:

1. Start the administrative console.
2. In the topology tree, expand **Resources** and click **JDBC Providers**.
3. Click the name of the provider for the data source that you want to configure.
4. Click **Data Sources**.
5. Click the name of the data source that you want to configure.
6. Click **Connection Pool**.
7. In the Min Connections field, specify a value. See Figure 8-23.
8. Click **Apply** or **OK**.
9. Save the configuration.
10. Stop and restart the application server.

Default value

The default value for the Minimum connection pool is 1.

Recommended value

Set the minimum pool size to handle the average load on the system. You can use Tivoli Performance Viewer to monitor the pool. See Figure 8-22 on page 259.

Avoiding deadlock

Deadlock can occur if the application requires more than one concurrent connection per thread, and the database connection pool is not large enough for the number of threads. Suppose each of the application threads requires two concurrent database connections and the number of threads is equal to the maximum connection pool size. Deadlock can occur when both of the following are true:

- ▶ Each thread has its first database connection, and all connections are in use.
- ▶ Each thread is waiting for a second database connection, and none become available, since all threads are blocked.

To prevent the deadlock in this case, the value set for the database connection pool must be at least one higher than the number of waiting threads to have at least one thread complete its second database connection.

To avoid deadlock, code the application to use, at most, one connection per thread. If the application is coded to require *C*, concurrent database connections per thread, the connection pool must support at least the following number of connections, where *T* is the maximum number of threads:

$$T * (C - 1) + 1$$

The connection pool settings are directly related to the number of connections that the database server is configured to support. If the maximum number of connections in the pool is raised, and the corresponding settings in the database are not raised, the application fails. Then Structured Query Language (SQL) exception errors are displayed in the SystemErr.log file.

Statement cache size

WebSphere Application Server provides a statement cache for each data source. This value represents the number of free prepared statements per connection in the connection pool. The statement cache stores query information for the data source.

The data source optimizes the processing of prepared statements to help make SQL statements process faster. It is important to configure the cache size of the data source to gain optimal statement execution efficiency. A prepared statement is a precompiled SQL statement that is stored in a prepared statement object. This object is used to efficiently execute the given SQL statement multiple times. If the JDBC driver specified in the data source supports precompilation, the creation of the prepared statement sends the statement to the database for precompilation. Some drivers may not support precompilation, and the prepared statement may not be sent until the prepared statement is executed.

If the cache is not large enough, useful entries are discarded to make room for new entries. In general, the more prepared statements your application has, the larger the cache should be. For example, if the application has five SQL statements, set the prepared statement cache size to 5, so that each connection has five statements.

Tivoli Performance Viewer can help tune this setting to minimize cache discards. Use a standard workload that represents a typical number of incoming client requests, use a fixed number of iterations, and use a standard set of configuration settings. Watch the PrepStmt Cache Discard counter of the JDBC Connection Pools module. The optimal value for the

statement cache size is the setting used to get either a value of zero or the lowest value for PrepStmt Cache Discards. See Figure 8-24.

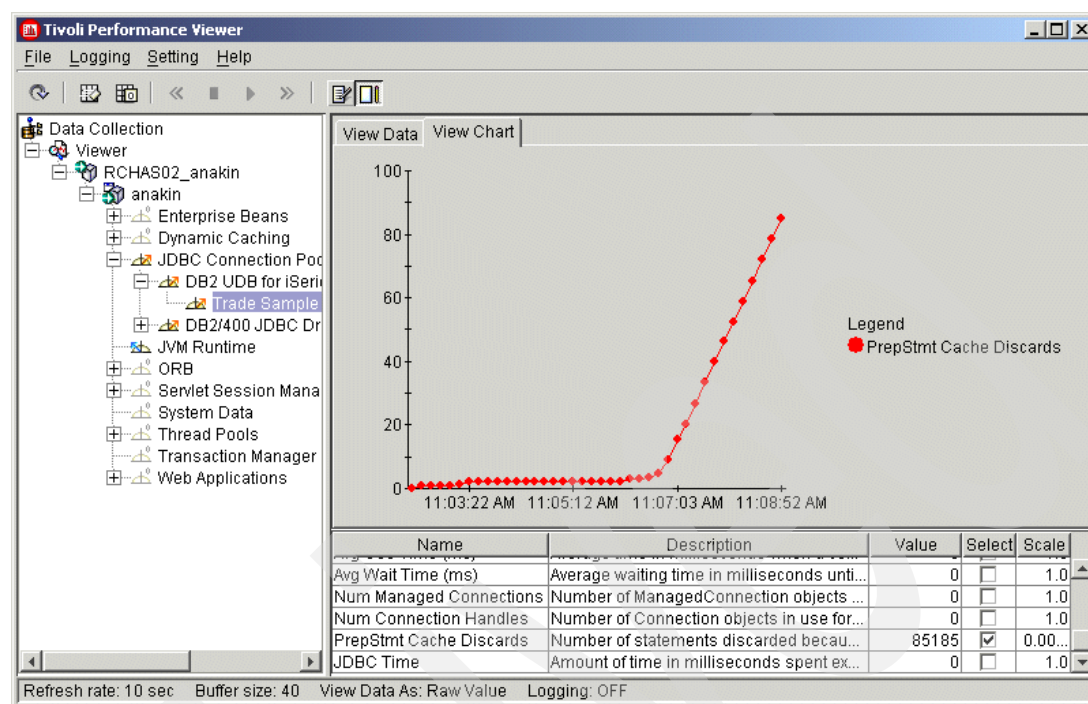


Figure 8-24 PrepStmt cache discards

As with the connection pool size, the statement cache size setting requires resources at the database server. Specifying a cache that is too large can impact database server performance. We recommend that you consult your database administrator to determine the best setting for the prepared statement cache size. See Table 8-5.

Table 8-5 Prepared statement: Testing with 100 clients and running it for 5 minutes

| Statement cache size | 10 | 15 | 20 |
|------------------------------|-------|-------|-------|
| PrepStmt Cache Discards | 88 | 52 | 2 |
| HTTP Requests | 15743 | 15824 | 15336 |
| Average Request Latency (ms) | 2.005 | 1.321 | 1.972 |

Note: The Statement cache size setting defines the maximum number of prepared statements cached per connection. This is different from WebSphere Application Server V4, where this was specified per container.

Setting Statement cache size

Follow these instructions:

1. Start the administrative console.
2. In the topology tree, expand **Resources** and click **JDBC Providers**.
3. Click the name of the provider for the data source that you want to configure.
4. Click **Data Sources**.
5. Click the name of the data source that you want to configure.
6. Scroll down. In the Statement Cache Size field, specify a value. See Figure 8-25.

[JDBC Providers](#) > [DB2 UDB for iSeries \(Native XA - V5R2 and later\)](#) > [Data Sources](#) >

TradeDataSource

Data Source is used by the application to access the data from the database. A data source is created under a JDBC provider which provides the specific JDBC driver implementation class.

Configuration

Test Connection

| General Properties | | |
|-------------------------------|---|---|
| Scope | * cells:RCHAS02B_anakin:nodes:RCHAS02B_anakin | The scope of the configured resource. This value indicates the configuration location for the configuration file. |
| Name | * TradeDataSource | The required display name for the resource. |
| JNDI Name | jdbc/TradeDataSource | The JNDI name for the resource. |
| Container managed persistence | <input checked="" type="checkbox"/> Use this Data Source in container managed persistence (CMP) | Enable if this data source will be used for container managed persistence of EJBs. This will cause a corresponding CMP connection factory which corresponds to this datasource to be created for the relational resource adapter. |
| Description | Trade3 DB2 JDBC Datasource | An optional description for the resource. |
| Category | | An optional category string which can be used to classify or group the resource. |
| Statement Cache Size | 100 statements | Number of free prepared statements per connection. This is different from the old datasource which is defined as number of free prepared statements per data source. |

Figure 8-25 Statement cache size setting

7. Click **Apply** or **OK**.
8. Save the configuration.
9. Stop and restart the application server.

Default value

The default value for Statement cache size is 10.

Recommended value

In most situations, we recommend that you set the Statement cache size value to the number of unique statements for each application that uses the data source. Setting the cache size to this value avoids cache discards and generally results in the best performance. However, if you set the cache size is too large, it may cause performance problems as a result of increased cache management and increased use of system resources. If you have a large number of unique statements, a smaller number may be appropriate.

Datasource recommendations

Adjust the QSQRVR prestart job settings for your system. On the iSeries server, JDBC access occurs via QSQRVR jobs.

- ▶ By default, five of these jobs are active initially.
- ▶ An application that establishes a large number of database connections over a short period of time may create those connections more quickly if these values are increased.
- ▶ Use the following command and increase the initial number of jobs, threshold, and additional number of jobs values:

```
CHGPJE SBSD(QSYS/QSYSWRK) PGM(QSYS/QSQRVR)
```

Only start the number of jobs required by the application, since there is some overhead associated with having QSQRVR jobs active, even if they are not being used.

For more information, see Chapter 4, “Tools for determining performance problems” on page 27.

8.2.5 EJB container

Each WebSphere Application Server includes an EJB container. You can use the following parameters to make adjustments that improve performance:

- ▶ Cache size
- ▶ Cleanup interval

Cache size

This parameter specifies the number of buckets in the cache. A *bucket* can contain more than one active enterprise bean instance, but performance is maximized if each bucket in the table has a minimum number of instances assigned to it. When the number of active instances within the container exceeds the cache size, the container periodically attempts to make some of the active instances passive ones. For the best balance of performance and memory usage, set this value to the maximum number of active instances expected during a typical workload.

Setting Cache size

To change or set this WebSphere Application Server value, follow these steps:

1. Start the administrative console.
2. In the topology tree, expand **Servers** and click **Application Servers**.
3. Click the name of the application server that you want to configure.
4. Click **EJB Container**.
5. On the EJB Container page, click **EJB Cache Settings**.
6. In the Cache Size field, specify a value. See Figure 8-27 on page 266.
7. Click **Apply** or **OK**.
8. Save the configuration.
9. Restart the application server.

Default value

The default value for Cache size is 2053.

Recommended value

For the best balance of performance and memory, set the Cache size value to the maximum number of active instances expected during a typical workload. To estimate the number of active instances, 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.

In our application, there are 6 entity beans, 100 transactions and 100 active sessions. The outcome of this is about 700 buckets. You can use the Tivoli Performance Viewer to monitor this setting. See Figure 8-26.

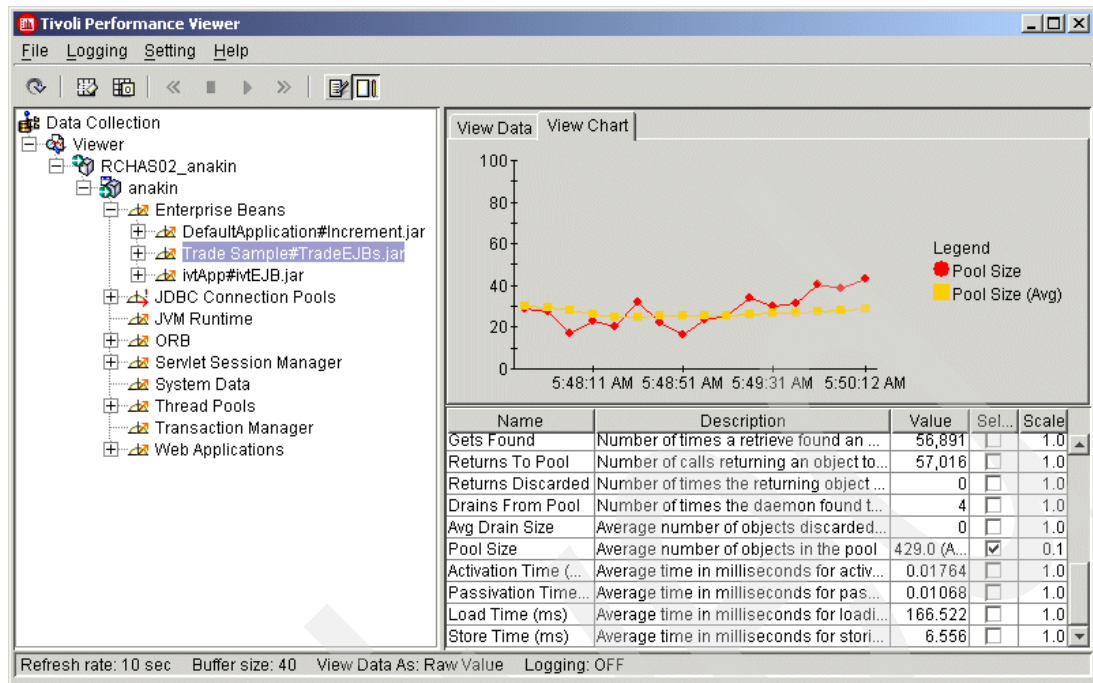


Figure 8-26 EJB container

The best result that we can achieve is with the setting of 1000 buckets. See Table 8-6.

Table 8-6 EJB Cache size test with 100 clients

| | First test | Second test | Third test | Fourth test |
|------------------------------|------------|-------------|------------|-------------|
| Cache Size | 2053 | 700 | 1000 | 3000 |
| HTTP Requests | 16090 | 15696 | 15624 | 15678 |
| Average Request Latency (ms) | 1.415 | 1.774 | 1.374 | 1.606 |

Inactive pool cleanup interval

The Inactive pool cleanup interval specifies the interval at which the container examines the pools of available bean instances to determine if some instances can be deleted to reduce memory usage.

Default value

The default value for the Inactive pool cleanup interval is 30000 milliseconds.

Cleanup interval

The Cleanup interval specifies the interval at which the container attempts to remove unused items from the cache. The cache manager tries to maintain some unallocated entries that can be allocated as needed. A background thread attempts to free some entries, while maintaining some unallocated entries. If the thread runs while the application server is idle, the application server does not need to remove entries from the cache before it can allocate new cache entries.

Setting Cleanup interval

To change or set this WebSphere Application Server value, follow these steps:

1. Start the administrative console.
2. In the topology tree, expand **Servers** and click **Application Servers**.
3. Click the name of the application server that you want to configure.
4. Click **EJB Container**.
5. On the EJB Container page, click **EJB Cache Settings**.
6. In the Cleanup interval field, specify a value. See Figure 8-27.

Application Servers > anakin > EJB Container >

EJB Cache Settings

Each EJB container keeps a cache of bean instances for ready access. This section allows you to specify the parameters to configure the cache. [i](#)

| Configuration | |
|--------------------|--|
| General Properties | |
| Cleanup interval | <input type="text" value="3000"/> milliseconds i The interval at which the container attempts to remove unused items from the cache in order to reduce the total number of items to the value of the cache size. |
| Cache size | <input type="text" value="2053"/> buckets i Specifies the number of buckets in the active instance list within the EJB container. |

Figure 8-27 EJB Cache Settings: Cleanup interval

7. Click **Apply** or **OK**.
8. Save the configuration.
9. Restart the application server.

Default value

The default value for Cleanup interval is 3000 milliseconds.

Recommended value

We recommend that you increase the value of parameter as the cache size increases.

8.2.6 Object Request Broker

An ORB uses the IIOP to manage the interaction between EJB clients and servers. It supports client requests and responses received from servers in a network-distributed environment. ORB tuning guidelines provide tips on using these parameters to tune the ORB. You can tune the following ORB parameters:

- ▶ Pass-by-reference
- ▶ Connection cache maximum
- ▶ Thread pool maximum size
- ▶ com.ibm.CORBA.ServerSocketQueueDepth

Monitoring the percentage of configured threads in use

Tivoli Performance Viewer shows a metric called *Percent Maxed*, which is used to determine how often the configured threads are used. A value that is consistently in the double-digits

indicates a possible bottleneck at the ORB. To remove the bottleneck, increase the number of threads.

Pass-by-reference

For remote method calls, this parameter specifies that the ORB passes parameters by reference instead of by value. If pass-by-reference is disabled, the ORB copies parameters to the stack before every remote method call is made.

Pass-by-value versus pass-by-reference

For EJB 1.1 beans, the EJB 1.1 specification states that method calls are to be pass-by-value. For every remote method call, the parameters are copied onto the stack before the call is made. This can be expensive. You can specify the pass-by-reference, which passes the original object reference without making a copy of the object. You can do this only if your application doesn't invoke an EJB in another JVM.

For EJB 2.0 beans, interfaces can be local or remote. For local interfaces, method calls are pass-by-reference, by default. For remote interfaces, you still have a choice of using pass-by-reference. Pass-by-reference can improve performance up to 50%.

Important: Test the application carefully with this setting before you decide whether to use it in a production environment.

The actual benefit observed is that if the EJB client and EJB server are installed in the same WebSphere Application Server instance, and the client and server use remote interfaces, specifying pass-by-reference can improve performance up to 50%.

When you try to adjust, pass-by-reference helps performance only in the case where non-primitive object types are being passed as parameters. Therefore, int and floats are always copied, regardless of the call model.

Attention: Pass-by-reference can be problematic and lead to unexpected results. If an object reference is modified by the remote method, the change may be seen by the caller.

Enabling Pass-by-reference

To enable this WebSphere Application Server setting, follow these steps:

1. Start the administrative console.
2. In the topology tree, expand **Servers** and click **Application Servers**.
3. Click the name of the application server that you want to configure.
4. Scroll down and click **ORB Service**.
5. On the ORB Service page, select the **Pass by reference** check box. See Figure 8-28.
6. Click **Apply** or **OK**.
7. Save the configuration.
8. Stop and restart the application server.

[Application Servers](#) > [anakin](#) >
ORB Service
 Configuration settings for the object request broker (ORB). ⓘ

Configuration

| General Properties | |
|--------------------|--|
| Request timeout | * 180 seconds ⓘ The number of seconds to wait before timing out on a request message. |
| Pass by reference | <input checked="" type="checkbox"/> ⓘ When enabled, this specifies that the ORB is to pass parameters by reference instead of by value, which bypasses a copy operation. Enable this property with caution, because unexpected behavior might occur. |

Apply OK Reset Cancel

| Additional Properties | |
|-----------------------------------|--|
| Thread Pool | Thread pool settings for the orb. |
| Custom Properties | Additional custom properties for this service which may be configurable. |

Figure 8-28 Pass-by-reference

Default value

Pass-by-value is the default for remote interfaces. Pass-by-reference is the default for EJB 2.0 local interfaces.

Recommended value

Enabling pass-by-reference can enhance performance, but is not specification compliant. Calls to EJB methods are normally made using Pass-By-Value call semantics, which makes copies of data passed to and from the methods. This is in accordance with the EJB specification.

However, when an EJB is called from another EJB or servlet in the same JVM (for example, the same application server), performance can be improved by passing the data by reference instead of copying it. The EJB Specification mandates Pass-By-Value semantics. While most applications should work properly with pass-by-reference enabled, some valid applications may not work correctly. Enterprise bean local interfaces can provide many of the performance benefits of pass-by-reference in a specification compliant manner.

Connection cache maximum

The Connection cache maximum parameter (com.ibm.CORBA.MaxOpenConnections) specifies the largest number of connections allowed to occupy the service's connection cache.

Setting Connection cache maximum

To change or set this WebSphere Application Server setting value, use these steps:

1. Start the administrative console.
2. In the topology tree, expand **Servers** and click **Application Servers**.
3. Click the name of the application server that you want to configure.
4. Scroll down and click **ORB Service**.

- On the ORB Service page, In the Connection cache maximum field, specify a value. See Figure 8-29.

[Application Servers](#) > [anakin](#) >
ORB Service
 Configuration settings for the object request broker (ORB). ⓘ

Configuration

| General Properties | | |
|--------------------------|-------------------|---|
| Request timeout | * 180 seconds | ⓘ The number of seconds to wait before timing out on a request message. |
| Request retries count | * 1 retries | ⓘ The number of times that the ORB attempts to send a request if a server fails. Retrying sometimes enables recovery from transient network failures. |
| Request retries delay | * 0 milliseconds | ⓘ The number of milliseconds between request retries. |
| Connection cache maximum | * 240 connections | ⓘ Largest number of connections allowed to occupy the service's connection cache. |

Figure 8-29 Connection cache maximum

- Click **Apply** or **OK**.
- Save the configuration.
- Stop and restart the application server.

Default value

The default value for Connection cache maximum is 240.

Recommended value

Adjust this value as necessary to support the number of clients that connect to the server-side ORB. You can increase this value to support the heavy load up to 1000 clients.

Thread pool maximum size

Thread pool maximum size specifies the maximum number of threads in the ORB pool. Each call to an EJB method from a separate JVM (either on the same system or another system) uses an ORB thread. If your application calls EJB methods only from servlets in the same JVM, you may not need to tune this parameter.

Setting Thread pool maximum size

To change or set this WebSphere Application Server value, follow these steps:

- Start the administrative console.
- In the topology tree, expand **Servers** and click **Application Servers**.
- Click the name of the application server that you want to configure.
- Scroll down and click **ORB Service**.
- On the ORB Service page, click **Thread Pool**.
- In the Maximum size field, specify a value. See Figure 8-30.
- Click **Apply** or **OK**.
- Save the configuration.
- Restart the application server.

[Application Servers](#) > [anakin](#) > [ORB Service](#) >

Thread Pool

A thread pool allows components of the server to reuse threads to eliminate the need to create new threads at runtime. Creating new threads is typically a time and resource intensive operation. [i]

Configuration

General Properties

| | | |
|---------------------------|---|--|
| Minimum Size | * 10 threads | [i] Specifies the minimum number of threads to allow in the pool. |
| Maximum Size | * 50 threads | [i] Specifies the maximum number of threads to allow in the pool. |
| Thread inactivity timeout | * 3500 milliseconds | [i] Specifies the number of milliseconds of inactivity that should elapse before a thread is reclaimed. |
| Is Growable | <input type="checkbox"/> Allow thread allocation beyond maximum thread size | [i] Specifies whether the number of threads can increase beyond the maximum size configured for the thread pool. |

Apply
OK
Reset
Cancel

Figure 8-30 ORB thread pool size

Default value

The default value for Thread pool maximum size is 50.

Recommended value

Increase this value if the Percent Maxed metric in Tivoli Performance Viewer is consistently greater than 10. In general, we recommend that you use closed queues. Make sure that the Is Growable property is not selected. Figure 8-31 shows the Percent Maxed value.

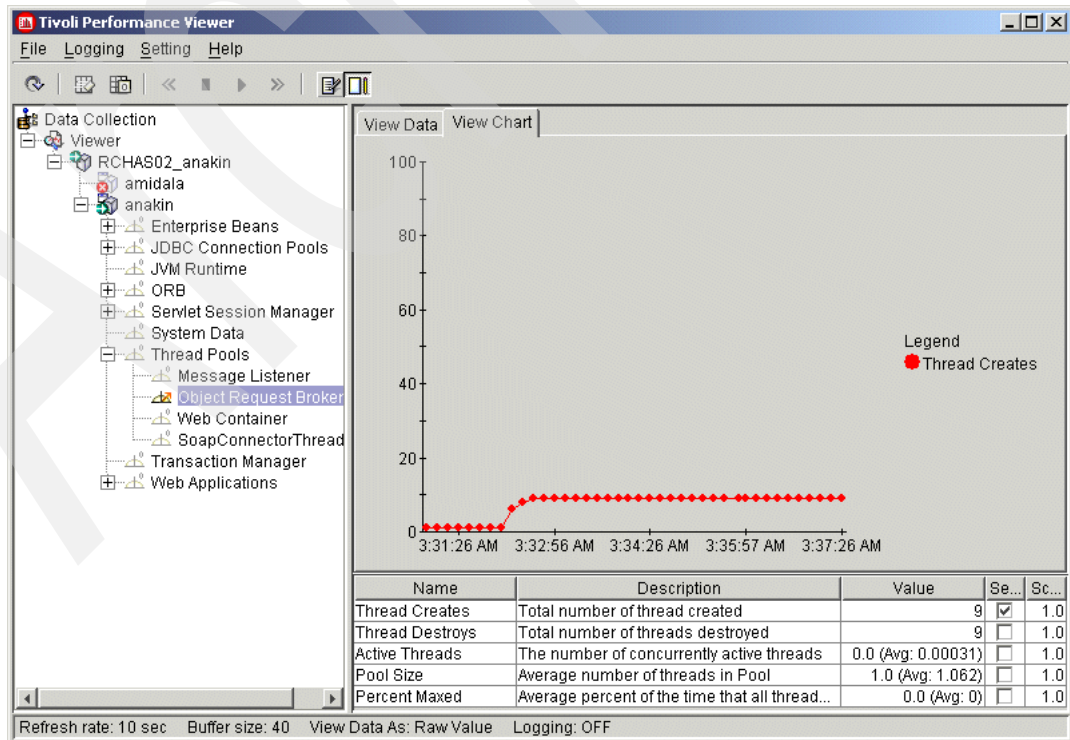


Figure 8-31 ORB thread pool size

com.ibm.CORBA.ServerSocketQueueDepth

The com.ibm.CORBA.ServerSocketQueueDepth property corresponds to the length of the TCP/IP stack listen queue. It prevents WebSphere Application Server from rejecting requests when there is no space in the listen queue. If several clients connect simultaneously to the server-side ORB, you can increase this parameter to support up to 1000 clients.

Setting com.ibm.CORBA.ServerSocketQueueDepth

To define this WebSphere Application Server setting, follow these steps:

1. Start the administrative console.
2. In the topology tree, expand **Servers** and click **Application Servers**.
3. Click the name of the server for which you want to set custom ORB properties.
4. Scroll down and click **ORB Service**.
5. On the ORB Service page, click **Custom Properties**.
6. To add a custom property, use these steps:
 - a. Click **New**.
 - b. Specify a name and value for the property. In this example (Figure 8-32), we specify com.ibm.CORBA.ServerSocketQueueDepth for Name and 50 for Value. You can also specify a description.

[Application Servers](#) > [anakin](#) > [ORB Service](#) > [Custom Properties](#) >

New

Specifies arbitrary name/value pairs of data, where the name is a property key and the value is a string value which can be used to set internal system configuration properties.

Configuration

| General Properties | | |
|--------------------|----------------------------------|---|
| Name | * n.CORBA.ServerSocketQueueDepth | Specifies the name (or key) for the property. |
| Value | * 50 | Specifies the value paired with the specified name. |
| Description | ServerSocketQueueDepth | Provides information about the name-value pair. |

Figure 8-32 ORB Service Custom Properties setting

- c. Specify the custom property or properties as shown in Figure 8-33 and click **OK**.

[Application Servers](#) > [kimAS02](#) > [ORB Service](#) >

Custom Properties

Specifies arbitrary name/value pairs of data, where the name is a property key and the value is a string value which can be used to set internal system configuration properties. [?](#)

Total: 8

☐ Filter

☐ Preferences

| <input type="checkbox"/> | Name | Value |
|--------------------------|---|---|
| <input type="checkbox"/> | com.ibm.CORBA.ConnectionInterceptorName | com.ibm.ISecurityLocalObjectBaseL13Impl.SecurityConnectionInterceptor |
| <input type="checkbox"/> | com.ibm.CORBA.FragmentSize | 1024 |
| <input type="checkbox"/> | com.ibm.CORBA.RasManager | com.ibm.websphere.ras.WsOrbRasManager |
| <input type="checkbox"/> | com.ibm.CORBA.ServerSocketQueueDepth | 50 |

Figure 8-33 ORB Custom Properties

7. Save the configuration.
8. Restart the application server.

Default value

The default value for `com.ibm.CORBA.ServerSocketQueueDepth` is 50.

Recommended value

There is no recommended value for `com.ibm.CORBA.ServerSocketQueueDepth`.

8.2.7 Additional performance tips

The following sections explain additional tuning knobs within WebSphere Application Server that can optimize the performance.

JNDI

Cache JNDI lookups in your application. You can use JNDI to look up any kind of object. The most common lookups are for datasource objects and EJB home objects. When possible, your application should do these lookups only once, and then cache the results for future use.

WebSphere does cache JNDI lookups automatically, but some additional overhead is involved when using this cache that you can avoid by caching the lookups in your application.

Logs, traces, and debugging

Be sure that you don't use the logs anymore. We recommend that you keep the SystemOut and SystemErr collection running. On a production system, you can disable the following values:

- In the WebSphere Application Server administrative console, select **Servers** → **Application Servers** → **server name** → **Web Container** → **ORB Service**. Then disable **ORB tracing**.

- ▶ In the WebSphere Application Server administrative console, select **Servers** → **Application Servers** → *server name* → **Web Container** → **Diagnostic Trace Service**. Then disable **Trace**.
- ▶ In the WebSphere Application Server administrative console, select **Servers** → **Application Servers** → *server name* → **Web Container** → **Debugging Service**. Then disable **Debugging at Startup**.
- ▶ In the WebSphere Application Server administrative console, select **Servers** → **Application Servers** → *server name* → **Web Container** → **Java Virtual Machine**. Then disable:
 - Verbose garbage collection
 - Verbose JNI
 - Debug Mode
- ▶ In the WebSphere Application Server administrative console, select **Troubleshooting** → **Logging and Tracing** → *server name* → **Process Logs**. Go to Stdout File Name and change it to "" (nothing is collected anymore).

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

- ▶ In the WebSphere Application Server administrative console, select **Servers** → **Application Servers** → *server name* → **Performance Monitoring Service**. Then select **Disable at Startup**.
- ▶ In the WebSphere Application Server administrative console, select **Troubleshooting** → **Logging and Tracing** → *server name* → **JVM Logs**. Change Stdout File Name to "" (nothing is collected anymore).
- ▶ In the WebSphere Application Server administrative console, select **Troubleshooting** → **Logging and Tracing** → *server name* → **IBM Service Logs**. Then disable **Service log**.
- ▶ In the WebSphere Application Server administrative console, select **Troubleshooting** → **WebSphere Configuration Problems**. Set Configuration Document Validation to **None: Do not validate documents**.
- ▶ In the WebSphere Application Server administrative console, select **Troubleshooting** → **Performance Monitoring Request Metrics**. Disable **Request Metrics**.

Attention: This step results in only minor performance improvements, while it significantly decreases the problem determination possibilities. If a problem occurs, there no information is available for debugging. In addition, the lack of error messages may hide a problem. For these reasons, we *do not recommend* this technique. Instead, use the technique in 9.5.7, “Minimizing or eliminating the use of System.out.println()” on page 311.

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.

Setting Class auto reload and reload

To change or set this WebSphere Application Server option, follow these steps:

1. Start the administrative console.
2. In the topology tree, expand **Applications** and click **Enterprise Applications**.
3. Click the name of the application that you want to configure.
4. Scroll down, deselect **Reload Enabled** or specify a value in the Reload Interval field. See Figure 8-34.

Enterprise Applications >
Trade3
Enterprise Applications ⓘ

Configuration Local Topology

| General Properties | | |
|-----------------------------|-------------------------------------|--|
| Name | * Trade3 | ⓘ Specifies a logical name for the application. Application names must be unique within a cell. |
| Starting Weight | * 1 | ⓘ Specifies the order in which applications are started. Applications with lower startup order are started before those with higher startup order. |
| Application Binaries | * \${APP_INSTALL_ROOT}/RCHAS02B | ⓘ Specifies the full path name of the enterprise application binary file. The path name can be an absolute path or can contain a pathmap variable such as APP_INSTALL_ROOT. |
| Use Metadata From Binaries | <input type="checkbox"/> | ⓘ Specifies whether the application server will use the binding, extensions, and deployment descriptors located with the application deployment document, the deployment.xml file (default), or those located in the application's ear file. |
| Enable Distribution | <input checked="" type="checkbox"/> | ⓘ Specifies whether the application will be distributed automatically to other nodes on the cell. The default is for automatic distribution. |
| Classloader Mode | * PARENT_FIRST | ⓘ Specifies whether classes are loaded via the parent classloader before this one. |
| WAR Classloader Policy | * Module | ⓘ Defines whether there is a single classloader for all WARs in the application or a classloader per WAR in the application. |
| Create MBeans For Resources | <input checked="" type="checkbox"/> | ⓘ Create MBeans For Resources |
| Reload Enabled | <input checked="" type="checkbox"/> | ⓘ Specifies if class reloading is enabled for application files when they get updated. |
| Reload Interval | 5 | ⓘ The timeperiod (in seconds) in which the application's filesystem will be scanned for updated files. |

Figure 8-34 Disabling auto reload

5. Click **Apply** or **OK**.
6. Save the configuration.
7. Stop and restart your application.

Pre-compile JSP during installation

If you want WebSphere Application Server to precompile JSP files as part of the installation, select Pre-compile JSP or specify the `-preCompileJSPs` option when you use the `wsadmin` tool. If you do not select this option, JSPs are compiled the first time they are accessed.

If you have a lot of JSPs in your application, use the JSP batch compiler.

Enabling Pre-Compile JSP

To enable this WebSphere Application Server option, follow these steps:

1. Start the administrative console.
2. In the topology tree, expand **Applications** and click **Install New Application**.
3. Specify the EAR, WAR, or JAR module to upload and install. Click **Next**.
4. Select your settings. Click **Next**.
5. Select the **Pre-compile JSP**. See Figure 8-35.

Install New Application

Allows installation of Enterprise Applications and Module

→ **Step 1: Provide options to perform the installation**

Specify the various options available to prepare and install your application.

| AppDeployment Options | Enable |
|----------------------------------|-------------------------------------|
| Pre-compile JSP | <input checked="" type="checkbox"/> |
| Directory to Install Application | <input type="text"/> |
| Distribute Application | <input checked="" type="checkbox"/> |
| Use Binary Configuration | <input type="checkbox"/> |
| Deploy EJBs | <input type="checkbox"/> |
| Application Name | <input type="text" value="Trade3"/> |
| Create MBeans for Resources | <input checked="" type="checkbox"/> |
| Enable Class Reloading | <input checked="" type="checkbox"/> |
| Reload Interval in Seconds | <input type="text" value="5"/> |
| Deploy WebServices | <input type="checkbox"/> |

Next Cancel

Figure 8-35 Selecting Pre-compile JSP

6. Select and define all settings for your application installation. During the installation, the JSP is compiled.
7. Save the configuration.
8. Start your application.

JSP batch compiler

WebSphere Application Server allows you to compile JSP files written to the JSP specification as a batch. This improves performance by reducing the response time on the first request. Previous versions of WebSphere Application Server compile JSP files only when the page is first requested. For iSeries, another option to consider is to use the Pre-touch tool, which can compile and load JSP class files into the WebSphere Application Server JVM for improved performance over the JSP batch compiler.

Product

The JSPBatchCompiler script is available in WebSphere Application Server.

Authority

To run this script, your user profile must have *ALLOBJ authority.

Running the JspBatchCompiler command

To use the batch compiler for JSP files, follow these steps:

1. On an iSeries command line, run the Start Qshell (STRQSH) command.
2. The Qshell command prompt \$ appears. Change your directory:

```
cd /QIBM/ProdData/WebAS51/Base/bin
```

3. Run the JspBatchCompiler command as shown in Example 8-4.

Example 8-4 JspBatchCompiler run

```
./JspBatchCompiler -enterpriseapp.name trade3 -webmodule.name trade3Web.war -  
instance anakin -server.name anakin
```

```
JSPG0005I: JspBatchCompiler: Initializing with:  
JSPG0006I: Config Root: /QIBM/UserData/WebAS51/Base/anakin/config.  
JSPG0007I: Cell: AS02B_anakin.  
JSPG0008I: Node: AS02B_anakin.  
JSPG0009I: Server: anakin.  
JSPG0010I: Enterprise Application: trade3.  
JSPG0011I: Web Module: trade3Web.war.  
JSPG0026I: JspBatchCompiler: Initializing server: anakin.  
JSPG0003I: Reading server configuration...  
JSPG0004I: Finished reading server configuration.  
JSPG0015I: WAS Install Root: /QIBM/ProdData/WebAS51/Base.  
JSPG0016I: User Install Root: /QIBM/UserData/WebAS51/Base/anakin.  
JSPG0017I: APP Install Root: /QIBM/UserData/WebAS51/Base/anakin/installedApps.  
JSPG0018I: Binaries URL:  
/QIBM/UserData/WebAS51/Base/anakin/installedApps/AS02B_anakin/trade3.ear.  
JSPG0019I: WebModule URL:  
/QIBM/UserData/WebAS51/Base/anakin/installedApps/AS02B_anakin/trade3.ear/trade3Web.war.  
JSPG0020I: Temp Directory:  
/QIBM/UserData/WebAS51/Base/anakin/temp/AS02B_anakin/anakin/trade3/trade3Web.war.  
JSPG0038I: Compile:  
/QIBM/UserData/WebAS51/Base/anakin/installedApps/AS02B_anakin/trade3.ear/trade3Web.war/WEB-  
INF.  
JSPG0038I: Compile:  
/QIBM/UserData/WebAS51/Base/anakin/installedApps/AS02B_anakin/trade3.ear/trade3Web.war/WEB-  
INF/wsd1.  
JSPG0038I: Compile:  
/QIBM/UserData/WebAS51/Base/anakin/installedApps/AS02B_anakin/trade3.ear/trade3Web.war/WEB-  
INF/classes.  
JSPG0038I: Compile:  
/QIBM/UserData/WebAS51/Base/anakin/installedApps/AS02B_anakin/trade3.ear/trade3Web.war/WEB-  
INF/classes/com.  
JSPG0038I: Compile:  
/QIBM/UserData/WebAS51/Base/anakin/installedApps/AS02B_anakin/trade3.ear/trade3Web.war/WEB-  
INF/classes/com/ibm.  
JSPG0038I: Compile:  
/QIBM/UserData/WebAS51/Base/anakin/installedApps/AS02B_anakin/trade3.ear/trade3Web.war/WEB-  
INF/classes/com/ibm/websphere.  
JSPG0038I: Compile:  
/QIBM/UserData/WebAS51/Base/anakin/installedApps/AS02B_anakin/trade3.ear/trade3Web.war/WEB-  
INF/classes/com/ibm/websphere/samples.  
JSPG0038I: Compile:  
/QIBM/UserData/WebAS51/Base/anakin/installedApps/AS02B_anakin/trade3.ear/trade3Web.war/WEB-  
INF/classes/com/ibm/websphere/samples/trade.  
JSPG0038I: Compile:  
/QIBM/UserData/WebAS51/Base/anakin/installedApps/AS02B_anakin/trade3.ear/trade3Web.war/WEB-  
INF/classes/com/ibm/websphere/samples/trade/web.  
JSPG0038I: Compile:  
/QIBM/UserData/WebAS51/Base/anakin/installedApps/AS02B_anakin/trade3.ear/trade3Web.war/WEB-  
INF/classes/com/ibm/websphere/samples/trade/web/prim.  
JSPG0038I: Compile:  
/QIBM/UserData/WebAS51/Base/anakin/installedApps/AS02B_anakin/trade3.ear/trade3Web.war/docs  
.
```



```
JSPG0038I: Compile:
/QIBM/UserData/WebAS51/Base/anakin/installedApps/AS02B_anakin/trade3.ear/trade3Web.war/imag
es.
JSPG0042I: Code generation successful for /account.jsp.
JSPG0042I: Code generation successful for /accountImg.jsp.
JSPG0042I: Code generation successful for /config.jsp.
JSPG0042I: Code generation successful for /displayQuote.jsp.
JSPG0042I: Code generation successful for /error.jsp.
JSPG0042I: Code generation successful for /marketSummary.jsp.
JSPG0042I: Code generation successful for /order.jsp.
JSPG0042I: Code generation successful for /orderImg.jsp.
JSPG0042I: Code generation successful for /PingJsp.jsp.
JSPG0042I: Code generation successful for /PingServlet2Jsp.jsp.
JSPG0042I: Code generation successful for /portfolio.jsp.
JSPG0042I: Code generation successful for /portfolioImg.jsp.
JSPG0042I: Code generation successful for /quote.jsp.
JSPG0042I: Code generation successful for /quoteImg.jsp.
JSPG0042I: Code generation successful for /register.jsp.
JSPG0042I: Code generation successful for /registerImg.jsp.
JSPG0042I: Code generation successful for /runStats.jsp.
JSPG0042I: Code generation successful for /tradehome.jsp.
JSPG0042I: Code generation successful for /tradehomeImg.jsp.
JSPG0042I: Code generation successful for /welcome.jsp.
JSPG0042I: Code generation successful for /welcomeImg.jsp.
JSPG0045I: Compiling...

JSPG0033I: Successfully Exiting the JSP Batch Compiler.
```

This command compiles all of the JSPs for the enterprise application called trade3. Here is the same command submitted as batch from the OS/400 command prompt:

```
SBMJOB CMD(QSH CMD('cd /mycompile;/QIBM/ProdData/WebASE/ASE5/bin/JspBatchCompiler
-enterpriseapp.name ibmorder -server.name MyInstance -instance MyInstance')) JOB(BATCHCMPL)
```

Note: If the names that are specified for these arguments are composed of two or more words separated by spaces, you must add quotation marks (") around the names.

Following compilation, the compiled .class files for the JSPs in the example Web module are found within the temp subdirectory of the instance. For the default instance, this is /QIBM/UserData/WebAS51/Base/default/temp. See Figure 8-36.

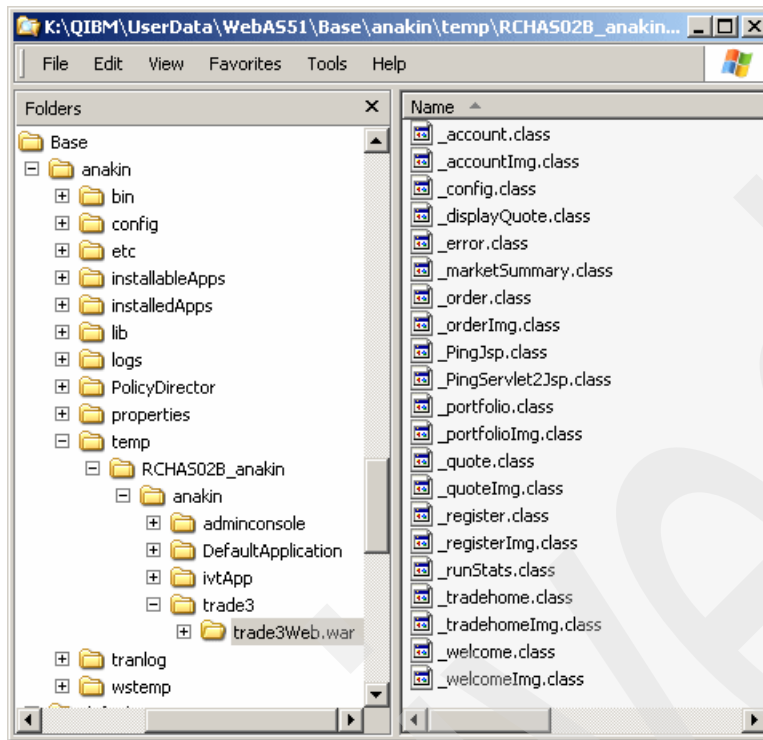


Figure 8-36 JspBatchCompiler compiled files

Disabling services you don't need

Look for the standard WebSphere Application Server services that your application doesn't use, for example:

- ▶ JMS Service

To disable the JMS Service, execute the following command in Qshell:

```
chgwasinst -instance yourInstanceName -embeddedjms no
```

- ▶ ORB (if you don't have a FAT client)

Change Threads Pool to 0.

8.3 Java Messaging Service tuning parameters

WebSphere Application Server supports asynchronous messaging as a method of communication based on the JMS programming interface. The base JMS support enables WebSphere enterprise applications to exchange messages asynchronously with other JMS clients by using JMS destinations (queues or topics). An enterprise application can explicitly poll for messages on a destination.

You can use the administrative console to tune JMS run-time components, resources, and the embedded messaging server.

Note: If you are not using JMS in your application server, we recommend that you disable it. See "Disabling services you don't need" on page 278. The JMS service consumes a low amount of resources during runtime, but can cause significant increases in the start and stop times for your application server.

8.3.1 Listener service

The message listener service is an extension to the JMS functions of the JMS provider. It provides a listener manager that controls and monitors one or more JMS listeners, which each monitor a JMS destination on behalf of a deployed message-driven bean.

Thread pool

The thread pool determines the maximum number of threads that the message listener service can run. Adjust this parameter when multiple message-driven beans are deployed in the same application server and the sum of their maximum session values exceeds the default value of 10.

Setting Thread pool

To change the Thread pool setting for WebSphere Application Server, use these steps.

1. Start the administrative console.
2. In the topology tree, expand **Servers** and click **Application Servers**.
3. Click the name of the application server that you want to configure.
4. Click **Message Listener Service**.
5. On the Message Listener Service page, click **Thread Pool**.
6. Edit the thread pool settings as needed (see Figure 8-37).

[Application Servers](#) > [trade51](#) > [Message Listener Service](#) >

Thread Pool

A thread pool allows components of the server to reuse threads to eliminate the need to create new threads at runtime. Creating new threads is typically a time and resource intensive operation. ⓘ

Configuration

| General Properties | | | |
|---------------------------|---|--------------|--|
| Minimum Size | * 10 | threads | ⓘ Specifies the minimum number of threads to allow in the pool. |
| Maximum Size | * 50 | threads | ⓘ Specifies the maximum number of threads to allow in the pool. |
| Thread inactivity timeout | * 3500 | milliseconds | ⓘ Specifies the number of milliseconds of inactivity that should elapse before a thread is reclaimed. |
| Is Growable | <input type="checkbox"/> Allow thread allocation beyond maximum thread size | | ⓘ Specifies whether the number of threads can increase beyond the maximum size configured for the thread pool. |

Apply OK Reset Cancel

Figure 8-37 Message listener service: Thread pool settings

7. Click **Apply** or **OK**.
8. Save the configuration.
9. Stop and restart the application server.

Default value

The default Minimum Size for thread pool is 10. The default Maximum Size value 50.

Recommended value

Set the minimum value to the sum of all message-driven beans maximum session values. Set the maximum to a value that is equal to or greater than the minimum value.

8.3.2 Listener port

A listener port is used to simplify administration of the association between a connection factory, destination, and deployed message-driven bean.

Maximum sessions

This parameter specifies the maximum number of concurrent JMS server sessions that a listener uses to process messages.

Setting Maximum sessions

To set this WebSphere Application Server option, follow these steps:

1. Start the administrative console.
2. In the topology tree, expand **Servers** and click **Application Servers**.
3. Click the name of the application server that you want to configure.
4. Click **Message Listener Service**.
5. On the Message Listener Service page, click **Listener Ports**.
6. Click the name of the listener port.
7. In the Maximum sessions field, specify a value. See Figure 8-38.

Application Servers > trade51 > Message Listener Service > Listener Ports > tradeport

Listener ports for Message Driven Beans to listen upon for messages. Each port specifies the JMS Connection Factory and JMS Destination that an MDB, deployed against that port, will listen upon. ⓘ

Runtime | **Configuration**

| General Properties | | |
|------------------------------|--|---|
| Name | * tradeport | ⓘ Name of the listener port |
| Initial State | * Started | ⓘ The execution state requested when the server is first started. |
| Description | Internal Listener Port for TradeBroker | ⓘ A description of the listener port, for administrative purposes |
| Connection factory JNDI name | * jms/TradeBrokerQCF | ⓘ The JNDI name for the JMS connection factory to be used by the listener port; for example, jms/connFactory1. |
| Destination JNDI name | * jms/TradeBrokerQueue | ⓘ The JNDI name for the destination to be used by the listener port; for example, jms/destn1. |
| Maximum sessions | 5 | ⓘ The maximum number of concurrent JMS server sessions used by a listener to process messages, in the range 1 through 2147483647. |
| Maximum retries | 10 | ⓘ The maximum number of times that the listener tries to deliver a message before the listener is stopped, in the range 0 through 2147483647. |
| Maximum messages | 1 | ⓘ The maximum number of messages that the listener can process in one JMS server session, in the range 0 through 2147483647. |

Figure 8-38 Listener Ports settings

8. Click **Apply** or **OK**.
9. Save the configuration.
10. Stop and restart the application server.

Default value

The default value for Maximum sessions is 1.

Recommended value

If you want to use *message concurrency* (multiple messages processed simultaneously), set the value to 2 to 4 sessions per system processor. Specify the lowest possible value to eliminate client thrashing. If you want a strict message order, set the value to 4. This value ensures that there is always a thread waiting in a hot state, but blocked on receiving the message.

8.3.3 JMS resources

A JMS provider enables asynchronous messaging based on the JMS. Application components can use JMS to communicate with each other.

XA enabled

An XA resource is a resource that supports the two-phase commit protocol. Two-phase commit allows applications to coordinate multiple resources, such as JMS queues, relational databases, JCA connectors, in a single transaction, called a Global Transaction or an XA transaction.

This attribute specifies whether the connection factory is for XA or non-XA coordination of messages. If you set this property to `NON_XA`, the JMS session is still enlisted in a transaction, but it uses the resource manager local transaction calls (`session.commit` and `session.rollback`) instead of XA calls. As a result, performance may improve. However, only a single resource can be enlisted in a transaction in WebSphere Application Server.

Enabling XA

To enable this WebSphere Application Server setting, follow these steps:

1. Start the administrative console.
2. In the topology tree, expand **Resources** and click **WebSphere JMS Provider** or **WebSphere MQ JMS Provider**. This parameter does not apply to generic JMS providers.
3. Click one of the links, depending on which provider you are using and which type of connection factory you want to configure:
 - **WebSphere Queue Connection Factories**
 - **WebSphere Topic Connection Factories**
 - **WebSphere MQ Queue Connection Factories**
 - **WebSphere MQ Topic Connection Factories**
4. Click the name of the connection factory that you want to configure.
5. Select or deselect **XA Enabled**. See the example in Figure 8-39.

[WebSphere JMS Provider](#) > [WebSphere Queue Connection Factories](#) >

TradeBrokerQCF

A queue connection factory is used to create connections to the associated JMS provider of JMS queue destinations, for point-to-point messaging. Use WebSphere Queue Connection Factory administrative objects to manage queue connection factories for the internal WebSphere JMS provider. [i](#)

Configuration

General Properties

| | | |
|-----------|---|---|
| Scope | * cells:AS05_trade51:nodes:AS05_trade51 | i The scope of the configured resource. This value indicates the configuration location for the configuration file. |
| Name | * TradeBrokerQCF | i The required display name for the resource. |
| JNDI Name | * jms/TradeBrokerQCF | i The JNDI name for the resource. |

| | | |
|------------|---|--|
| XA Enabled | <input checked="" type="checkbox"/> Enable XA | i Attribute to indicate whether or not the JMS provider is XA enabled or not. This attribute only applies to specialized models of JMSConnectionFactory. It is meaningless for GenericJMSConnectionFactories, as they define such feature enablements through name/value property pairs. |
|------------|---|--|

Figure 8-39 XA Enabled setting

- Click **Apply** or **OK**.
- Save the configuration.
- Stop and restart the application server.

Default value

By default, XA is enabled.

Recommended value

Do not enable XA when the message queue or topic received is the only resource in the transaction. Enable XA when other resources, including other queues or topics, are involved.

Connection pool size

The connection pool size specifies the maximum number of connections that the pool can contain. This parameter does not apply to generic JMS providers.

Setting the connection pool size


To set this WebSphere Application Server option, follow these steps:

- Start the administrative console.
- In the topology tree, expand **Resources**.
- Navigate through one of these paths to the connection factory that you want to configure:
 - WebSphere JMS Provider** → **WebSphere Queue Connection Factories** → *conn_factory_name* → **Connection Pool**
 - WebSphere JMS Provider** → **WebSphere Topic Connection Factories** → *conn_factory_name* → **Connection Pool**
 - WebSphere MQ JMS Provider** → **WebSphere MQ Queue Connection Factories** → *conn_factory_name* → **Connection Pool**
 - WebSphere MQ JMS Provider** → **WebSphere MQ Topic Connection Factories** → *conn_factory_name* → **Connection Pool**

4. In the Max Connections field, specify a value. See the example in Figure 8-40.

[WebSphere JMS Provider](#) > [WebSphere Queue Connection Factories](#) > [TradeBrokerOCF](#) >

Connection Pools

Connection pool properties that can be modified to change the behavior of the J2C connection pool manager. Default values are provided for non-production use. Reviewing and possible modification of these configuration values is recommended. 

Configuration






| General Properties | | |
|--------------------|---|---|
| Scope | cells:RCHAS02B_anakin:nodes:RCHAS02B_anakin |  The scope of the configured resource. This value indicates the configuration location for the configuration file. |
| Connection Timeout | <input type="text" value="180"/> seconds |  Interval, in seconds, after which a connection request times out and a <code>ConnectionWaitTimeoutException</code> is thrown. |
| Max Connections | <input type="text" value="10"/> connections |  The maximum number of <code>ManagedConnections</code> that can be created in this pool. |
| Min Connections | <input type="text" value="1"/> connections |  The minimum number of <code>ManagedConnections</code> that should be maintained. |
| Reap Time | <input type="text" value="180"/> seconds |  Interval, in seconds, between runs of the pool maintenance thread. |

Figure 8-40 Connection pools setting

5. Click **Apply** or **OK**.
6. Save the configuration.
7. Stop and restart the application server.

Default value

The default value for Max Connections is 10.

Recommended value

For the connection pool size, specify a value lower than the value for the Max Connections option in the Web container. Lower settings, such as 10 to 30 connections, perform better than higher settings, such as 100.

8.4 Tuning your security configuration

Performance issues typically involve trade-offs between function and speed. Usually, the more function and the more processing that are involved, the slower the performance is. Consider the type of security that is necessary and what you can disable in your environment. For example, if your application servers are running in a Virtual Private Network (VPN), consider whether you must disable Single Sockets Layer (SSL). If you have a lot of users, decide whether you can map them to groups and associate the groups to your J2EE roles. You must consider these items when designing your security infrastructure.

There is always a trade-off between performance, features, and security. Security typically adds more processing time to your requests, but for a good reason. Not all security features are required in your environment. When you decide on tuning security, create a benchmark before you make any changes to ensure that the change is improving performance.

In a large scale deployment, performance is important. Running benchmark measurements with different combinations of features can help you to determine the best performance versus benefit configuration for your environment.

The following list provides tips on what to look for while securing your WebSphere Application Server instance:

- ▶ Only turn on security if it is required for the application. Enabling security has a substantial performance impact.
- ▶ There are two security types: SSL and WebSphere Application Server Global Security.
- ▶ Security is a global setting. When security is enabled, performance can be decreased between 10% and 20%. Therefore, disable security when you do not need it.
- ▶ Fine-tune the security cache time out for the environment. If WebSphere Application Server security is enabled, the security cache time out can influence performance. The time out parameter specifies how often to refresh the security-related caches.

The actual benefit observed in a 20 minute performance test, when the cache time out was set so that a time out did not occur, was a performance improvement of 40%.



Java and WebSphere application design

This chapter is full of tips that to help you to build your application to perform better. The tips include examples when possible.

Normally all of these tips and techniques are more noticeable when the system's load is heavy. In these cases, your application needs to perform better and should have a better scalability.

Note: WebSphere Development Studio Client for iSeries Advanced Edition V5.1 is the development tool that is used for all examples in this chapter.

9.1 Java general design

There is a lot you can do within the Java code. Since these techniques use general Java code, they are useful for any Java code, even if you are running stand-alone applications, servlets, JavaServer Pages (JSPs), or Enterprise JavaBeans (EJBs) in WebSphere.

9.1.1 String manipulation

One of the most used classes in any Java application is the `String`. `String` is a special object in Java. It has the operators, such as `+` and `+=`, which make it easy to use `String` objects. However, some `String` manipulations have a high cost.

String creation

The Java virtual machine (JVM) maintains an internal list of all the used `Strings`. When a new `String` is created, the JVM tries to match it with any existing `String` and points to it. You can create `Strings` in two ways:

► First method

```
String s1 = "Rochester";  
String s2 = "Rochester";
```

► Second method

```
String s1 = new String("Rochester");  
String s2 = new String("Rochester");
```

In the first method, only one `String` is created. In the second method, two `Strings` are created. The application functionality is the same in both cases, but the first one performs much better.

We run the code in Example 9-1 to show the difference.

Example 9-1 String creation

```
public void StringCreation1(long l)  
{  
    for (int i = 0; i < l; i++) {  
        for (int j = 0; j < 25; j++) {  
            String s1 = "Rochester";  
            String s2 = "Rochester";  
        }  
    }  
}  
  
public void StringCreation2(long l)  
{  
    for (int i = 0; i < l; i++) {  
        for (int j = 0; j < 25; j++) {  
            String s1 = new String("Rochester");  
            String s2 = new String("Rochester");  
        }  
    }  
}
```

In this example, we obtain the results shown in Table 9-1.

Table 9-1 String Creation test results

| Method | Time to perform 100000 iteration |
|-----------------|----------------------------------|
| StringCreation1 | 0 ms |
| StringCreation2 | 631 ms |

String versus StringBuffer

Many times in your application, you must concatenate Strings. Strings are immutable objects, which means that they cannot be changed. When you create a String object and change it, since the String cannot be modified, then another String object is created with a new value as shown in the following example:

```
String s1 = "hello";  
s1 = s1 + " to everybody";
```

The second line creates a second String object instead of changing the s1 String. When you do this frequently, it becomes a lot of work for the garbage collector. It is better to use the StringBuffer object for this type of operations. StringBuffer creates a char array to hold the String. It can also append other Strings much faster and with less usage of memory:

```
StringBuffer sb = new StringBuffer();  
sb.append("hello");  
sb.append(" to everybody");  
String s1 = sb.toString();
```

You take advantage of this when you deal with big Strings and the workload is heavy, which is mostly when you need better performance. Big Strings are used often in JSPs when you build the response for the user. In our test, StringBuffer is up to six times faster than String concatenation. A String's length is an important factor on performance in this case.

We run the code in Example 9-2 to show the difference.

Example 9-2 String test

```
public void StringBuffer(String s, long l)  
{  
    for (int i = 0; i < l; i++) {  
        StringBuffer sb = new StringBuffer();  
        for (int j = 0; j < 25; j++) {  
            sb.append(s);  
        }  
        String se = sb.toString();  
    }  
}  
  
public void String(String s, long l)  
{  
    for (int i = 0; i < l; i++) {  
        String st = new String();  
        for (int j = 0; j < 25; j++) {  
            st = st + s;  
        }  
    }  
}
```

We obtain the results shown in Table 9-2.

Table 9-2 String test result

| Method | Time to perform 100.000 iteration s.length() = 25 |
|--------------|--|
| StringBuffer | 1.923 ms |
| String | 7.191 ms |

String initialization in CompileTime or in RunTime

Let's say that you initialize a String with this code:

```
String s = "Train " + "08876" + " to " + "Madrid";
```

Then the compiler makes it as though you coded:

```
String s = "Train 08876 to Madrid";
```

This initialization is done during the compilation and does not impact the application execution. But if any of the Strings is a variable, then the compiler can't optimize it. Therefore, you don't write this line:

```
String s = "Train " + trainCode + " to " + city;
```

Instead you write this line:

```
String sb = (new StringBuffer()).append("Train ").append(trainCode).  
append(" to ").append(city).toString();
```

9.1.2 Object instantiation

Instantiation is the creation of a new object. When you create an object, some memory is needed to allocate the object itself and all the data stored in it. This is an expensive process in performance terms. Of course instantiation results in garbage collection to remove the objects when they are not referenced any more. The more complex an object is, the more overhead it takes to instantiate it.

The following sections provide tips regarding object instantiation.

Instantiation results in garbage collection

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;  
else  
    securityManager.reset();
```

This way, the securityManager object is created only if it does not already exist. reset() is a method that a user has to write.

Inherited chains too long and objects too complex

When you create an object, the constructor is called, so the object is initialized correctly and all the variables are stored in the heap. If the object is a subclass, the constructor of the

superclass is also called to initiate the superclass variables and store them, and so on. If the inherited chain is too long, there are a lot of tasks to do when you create an object.

This also applies for very complex objects. Sometimes you have an object that performs so many functions that it has a lot of initializing work. If this is a commonly created object, this will impact performance. You can try to split the object into several pieces and put all the related data together.

Parsing numbers

When you need to parse a String into a data primitive, you can use the static method for doing this instead of creating an object and using the instance method as you can see in the following sample.

That is, you don't use this string:

```
int myInt = new Integer(s).intValue();
```

Instead, you use this string:

```
int myInt = Integer.parseInt(s);
```

The first example uses the static method so it does not require an object instantiation. Consider Example 9-3.

Example 9-3 Parsing integers test

```
public void parseIntStatic(String s, long l){
    for (int i = 0; i < l; i++) {
        for (int j = 0; j < 25; j++) {
            int myInt = Integer.parseInt(s);
        }
    }
}

public void parseInt(String s, long l){
    for (int i = 0; i < l; i++) {
        for (int j = 0; j < 25; j++) {
            int myInt = new Integer(s).intValue();
        }
    }
}
```

More tips

Here are some tips in regard to object instantiation:

- ▶ Avoid instantiating objects in a loop.
- ▶ Reuse objects when possible.
- ▶ Set objects to null when they are no longer required so that the garbage collector can handle it as soon as possible.
- ▶ Use local variables instead of class variables when possible.
- ▶ Use data primitives (int), which are better than wrapper classes (Integer), when possible.

9.1.3 Loops

Applications use plenty of loops. You can save a lot of time and resources when you write good performing loops. There are some techniques that you can apply.

Using constant values to terminate loops

To check if a loop is going to finish, place a constant call instead of a method call. Otherwise every time the loop is iterating, this method call is executed. You can do this when you know the number of times the loop is going to iterate and the result of the method call is not changed during the loop execution. Normally this is true for the *for* loops.

That is, don't use this statement:

```
for (int j = 0; j < s.length(); j++) {...}
```

Instead, use this one:

```
int max = s.length();  
for (int j = 0; j < max; j++){...}
```

Termination condition

Each time the loop iterates, the termination condition is evaluated. If the condition is complex, you can make it faster using a short circuit.

- ▶ If you are using the && operator, place the most frequently false part on the left side of the condition.
- ▶ If you are using the || operator, place the most frequently true part on the left side of the condition.

Additional tips

Here are some other tips:

- ▶ Only put into the loop the code that is really needed to execute in each iteration.
- ▶ Don't instantiate objects into loops if possible.
- ▶ Use temporary variables for values that don't change during the loop.
- ▶ Avoid method calls into the loop.
- ▶ Use *int* data type as a loop index.

9.1.4 Exceptions

When an exception is thrown, a copy of the stack is taken so that it can be used during the catch block. The code shouldn't throw exceptions for controlling the application normal path. The exception should be thrown when something exceptional happens.

```
try {  
    ...  
} catch (Exception e) {  
    ...  
    e.printStackTrace();  
}
```

The use of try and catch blocks does not affect the performance if no exception is thrown. If you are running performance tests, you have to go through these exceptions in your tests to understand how your application will run in real life.

9.1.5 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 overhead 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. It 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: This results 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 with OPTIMIZE(40) performs method resolution without requiring the `java -O` option.

9.1.6 Logging

Normally logging is an important part of the application. It's frequently used during development and quite valuable when troubleshooting an application. However, logging is expensive in performance terms because it uses a lot of input/output (I/O) resources.

Instead of using `System.out.println(...)` directly in your program, you can use a class for logging where you can switch on logging, depending on the severity of the information to log. Example 9-4 provides a sample code.

Example 9-4 The EasyLog class

```
public class EasyLog {

    public static int globalMinSeverity = 11;

    public static final int DEVELOPMENT = 00;
    public static final int INFO = 10;
    public static final int WARNING = 40;
    public static final int STOP = 80;

    public static void writeLog(int severity, String thelog){
        if (severity >= globalMinSeverity)
            System.out.println(thelog);
    }

    public static void setGlobalSeverity(int severity){
        globalMinSeverity = severity;
    }
}
```

To use this class, initialize the severity level desired and then call to the `writeLog` method instead of `System.out.println`:

```
EasyLog.setGlobalSeverity(EasyLog.INFO);
...
EasyLog.writeLog(EasyLog.INFO, "Test Running...");
```

You can change the Global Severity at any moment during the runtime to switch on logging.

In JDK 1.4, a new logging application programming interface (API) is introduced, the *java.util.logging* package. You can use it for your logging. This API is designed to perform well.

9.1.7 Variables

If a constant is needed in your class, make it `static final`, so that only one copy of this constant is located in memory.

```
public static final int INFO = 10;
```

Normally local variables are faster than global variables, and faster than accessor methods too. Be careful and don't create problems with your application trying to make all your variables public. From a designing point of view, object variables should be private.

In the `doVariable` method in Example 9-5, it is faster to use `mylocal` or `myargument` than to use `myinstancevar`. This doesn't affect performance much.

Example 9-5 Variable scope

```
public class Variables {

    public static final int myConstant = 1;

    private int myinstancevar = 1;

    public Variables() {
        super();
    }

    public void doVariable(int myargument){
        int mylocal = 1;
    }

    private final int getMyinstancevar() {
        return myinstancevar;
    }
}
```

9.1.8 Collections

The Java 2 platform includes the new Collections Framework. A *collection* is an object that holds a group of objects you want to have together, so you can store and manipulate them as a single unit. There are collections for several purposes, so you can choose the correct one each time for design and performance purposes. The new framework provides more functionality and enhances the performance of the old collections.

Table 9-3 shows the general purpose implementations in the Collections Framework. These implementations support all the optional operations in the collection interfaces, and have no restrictions on the elements that they may contain. They are unsynchronized. However, there are static factories called *synchronization wrappers* that can add synchronization if needed.

Table 9-3 Java 2 Collection Framework

| Interfaces | Implementations | | | |
|------------|-----------------|-----------------|--------------|-------------|
| | Hash table | Resizable array | Balance tree | Linked list |
| Set | HashSet | | TreeSet | |
| List | | ArrayList | | LinkedList |
| Map | HashMap | | TreeMap | |

When you use these collections, remember these points:

- ▶ The Collection interface is a group of objects, with duplicates allowed.
- ▶ Set extends the Collection but forbids duplicates.
- ▶ List extends Collection as well, but allows duplicates, and introduces positional indexing.

To choose the correct Collection, follow these guidelines:

- ▶ Sets
 - Sets are slow in general.
 - Usually Set is the fastest set.
 - TreeSet is the slower set but provides iteration of keys in order.
- ▶ Lists
 - Usually ArrayList is the fastest list.
 - If the list is large and elements are frequently deleted or inserted, LinkedList is faster.
 - Stack has the same speed as Vector and provides last in, first out (LIFO) queue functionality.
- ▶ Maps
 - Usually HashMap is the fastest map.
 - TreeMap is the slower Map but provides iteration of keys in order.

One final tip regarding performance is to use iterators whenever you can to access the elements of a list, rather than using positional access.

Attention: Use care when using Collections because they may become the beginning of memory leaks issues. If you add elements to a Collection and never delete these references, this becomes a memory leak. Imagine an application with a Collection variable in the servlet to insert the customers that login the application. After a while, you have a list of all the customers that logged in the application, but if you never delete these customers from the Collection, it will grow endlessly.

9.1.9 Java Native Interface

Native method invocation on an iSeries server may not perform as well as native method invocation on other platforms. Java on the iSeries server has been optimized by moving the JVM below the machine interface (MI). Native method invocation requires a call to above MI code and may require expensive Java Native Interface (JNI) calls back into the JVM. Native methods should not carry out small routines, which you can easily write in Java. Only use native methods to start system functions that are relatively long running and are not available directly in Java.

9.1.10 Thread creation

The iSeries system kernel thread implementation is efficient. Thread creation is considered a lightweight operation. Threads within a process can reuse 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 multitasking heritage, unnecessary thread creation 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.

9.2 Accessing system services: Database operations

In a typical commercial environment, there are two major areas where processing overhead is incurred. One area is in running the Java program instructions, as explained 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 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 9-1 shows various ways to access the relational database or existing programs on the iSeries server.

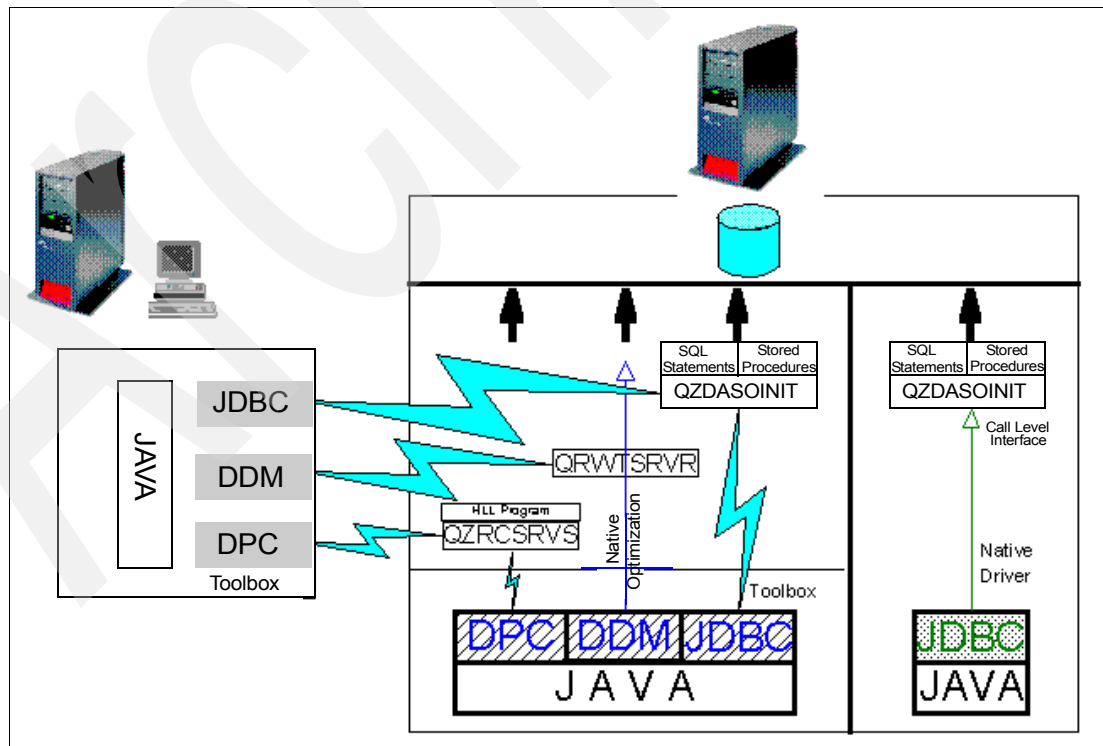


Figure 9-1 IBM Toolbox for Java system access

In Figure 9-1, 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.

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

- ▶ The DPC model is designed for client-server work, so the Java application communicates through the QZRC SRVS 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 done locally, then 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 9-1 shows a server Java application using the *native* JDBC driver.

Regardless of which driver is used, JDBC can access DB2 through Structured Query Language (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.

For further information, refer to DB2 Universal Database (UDB) for iSeries home page:

<http://www.iseries.ibm.com/db2>

9.2.1 IBM Toolbox for Java driver versus iSeries Developer Kit for Java driver

As described previously, two JDBC drivers are available when running a Java application on the iSeries server. One is the native driver or the IBM 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.

You can learn about other differences on the Web at:

<http://www-1.ibm.com/servers/enablsite/java/jdbc/jdbcfaq.html>

IBM Toolbox for Java JDBC 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.

Make sure that there is an adequate number of database server jobs ready to service client requests. Otherwise the system needs to start new instances of QZDASOINIT job that unnecessarily consumes system resources and slows down the connection time. To view the current settings for the QZDASOINIT prestart job, enter the following command:

```
DSPSBSD SBSD(QUSRWRK)
```

Select option 10 (Prestart job entries). Then, select option 5 on the QZDASOINIT program to display the details.

You can use the following command to change the parameters:

```
CHGPJE SBSDB(QUSRWRK) PGM(QSYS/QZDASOINIT)
```

For more information about tuning the server jobs used for the database access, see Chapter 4, “Tools for determining performance problems” on page 27.

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 Toolbox the Extended Dynamic mode is false, by default. Set the extended dynamic property to *true* to take advantage of the performance benefits with this feature.

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 they access the database. The result is additional latency and processing overhead.

iSeries Developer Kit for Java driver performance

The iSeries Developer Kit for Java driver 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 similar to 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 *remote* connection. This is not a hard limit for the local connections. In the unlikely situation that more than 500 are required, another connection can be established.

9.2.2 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 prestarted job named QSQSRVR. When Java establishes a connection to the DB, a message is sent to the BCI job log similar to this example:

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

When you analyze the database, monitor the QSQSRVR job for optimizer messages. You do this by placing the job in service mode and debug mode. That is you use the Start Service Job (STRSRVJOB) command, followed by the Start Debug (STRDBG) command. The job log then records the optimizer messages.

You can also use the Database Monitor to monitor this job (using the Start Database Monitor (STRDBMON) and End Database Monitor (ENDDBMON) or the Start Performance Monitor (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 to use the SQL Monitor available in Operations Navigator since V4R4. See Chapter 4, “Tools for determining performance problems” on page 27.

Statement cache for dynamic SQL

You can issue three kinds of SQL statements on the iSeries server:

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

Static SQL is the most efficient and supported SQL in IBM Developer Kit for Java. 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, or create an access plan for every execution of the statement.

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 handle parameter marker replacement. Not only are the statement strings stored, but other internal structures, such as access plans, are also stored. This reduces the need to recreate them each time the statement executes.

Database tuning

In an online transaction processing environment (OLTP), the query runtime should be short. You should then tune the database 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, use the debug feature or DB Monitor 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 that you can monitor and analyze optimizer messages are to place 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.

9.2.3 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 you issue requests from a client to the server.

Switching from one driver to another is a simple change that usually affects only one class or datasource configuration in WebSphere. 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.

9.2.4 Coding considerations with JDBC

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

Using the data source

The opening of a connection to the database is a quite expensive operation in terms of performance. At the same time, if the application doesn't close the connections, all the available connections will soon be busy and you will receive an exception. The best way to handle this is to use a connection pool that is an object that manages all the connections objects to DB2. On WebSphere, you can use a data source to manage your connections to the database.

Example 9-6 shows a class that you can use to create the connections using a data source.

Example 9-6 Data source Connection class

```
package com.ibm.rch.performance.java.db;

import java.sql.*;
import java.util.*;
import javax.naming.*;
import javax.sql.*;

public class PoolDB
{
    private java.lang.String dataSource;

    public PoolDB(String dSAux)
    {
        dataSource = dSAux;
    }

    public Connection connect() throws Exception, SQLException
    {
        DataSource ds = null;
        Hashtable parms = new Hashtable();
        parms.put(Context.INITIAL_CONTEXT_FACTORY,
            "com.ibm.websphere.naming.WsnInitialContextFactory");
        Context ctx = new InitialContext(parms);
        ds = (DataSource)ctx.lookup(this.dataSource);
        Connection con = ds.getConnection();
        return con;
    }
}
```

By using a class such as the one in Example 9-6, you can change your connections to the database whenever you want without changing all the application. You only need to change a datasource configuration.

Then to use this connection, you can use the code shown in Example 9-7.

Example 9-7 Using an SQL statement with a data source

```
public String doDb() throws SQLException{

    StringBuffer sb = new StringBuffer();
    PoolDB pool = new PoolDB("jdbc/myds");
    Connection con = null;
    Statement st = null;
    try {
        con = pool.connect();
        String sql = "Select * from employee";
        st = con.createStatement();
        ResultSet rs = st.executeQuery(sql);

        while (rs.next())
        {
            sb.append(rs.getString("EMPLNO"));
            sb.append(":");
        }
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    finally { //Important: allways close the connection
        if (st != null) st.close();
        if (con!=null) con.close();
    }

    return sb.toString();
}
```

Using prepared statements

It is good to improve performance by using prepared statements. When you execute an SQL statement in your application, the database has to perform several tasks before it starts executing the SQL. For example, it has to parse the SQL code to see if it is correct, look for the indexes to use during the execution, look for the fields that are needed, and so on. These tasks are known as the *access plan*. Creating a plan requires a lot of work.

When you use the Statement class, the access plan is thrown away after you run your SQL statement. This is desirable when you are using this statement only once because your server doesn't spend time storing the plan. In any normal application, you are using the same set of SQL statements many times. We recommend that you use PreparedStatement instead of Statement. System-wide cache or *SQLPKG is used.

Example 9-8 shows the code of using PreparedStatement.

Example 9-8 Use of PreparedStatement

```
Connection con = pool.connect();
String sentence = "Select * from employee where EMPLNO = ?";
PreparedStatement ps = con.prepareStatement(sentence);
for (int i = 0; i < 100; i++) {

    ps.setString(1, Integer.toString(i));
    ResultSet rs = ps.executeQuery();

    while (rs.next())
    {
        sb.append(rs.getString("EMPLNO"));
        sb.append(":");
    }
    rs.close();
    ps.close();
}
con.close();
```

We run the code in Example 9-9 to show the difference.

Example 9-9 Prepared Statement test

```
for (int i = 0; i < 1; i++) {
    String sentence = "Select * from LUIS.TRCUSTOMER where CUSTID = ?";
    PreparedStatement ps = con.prepareStatement(sentence);
    ps.setString(1, Integer.toString(i));
    ResultSet rs = ps.executeQuery();

    rs.close();
    ps.close();
}

for (int i = 0; i < 1; i++) {
    String sentence = new StringBuffer().append("Select * from LUIS.TRCUSTOMER where CUSTID
= ").append(i).append("'").toString();
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery(sentence);

    rs.close();
    stmt.close();
}
```

We obtain the results shown in Table 9-4.

Table 9-4 Prepared Statements test results

| Using... | Time to perform 500 iterations |
|-------------------|--------------------------------|
| PreparedStatement | 1.452 ms |
| Statement | 4.236 ms |

Retrieving only the necessary columns

It is common to write SQL statements using the syntax:

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

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

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

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

Using 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. On some platforms, using a column index requires the get to be performed in column sequence. This is not a restriction on the iSeries server.

Using a collection to obtain data

For example, you can retrieve several rows from a table to perform some task on each row. Sometimes these tasks take a long time. In this case, you use a connection that is locked for another user. In such a case, it is better to obtain all the data from the table and create a collection with all this data stored. Then close the connection and start performing the long task that you want to do from the data in the collection. By using this technique, the connection is available for another user faster.

Example 9-10 shows how you can use Collection.

Example 9-10 Using Collection to obtain the data from the database

```
// Using a Collection
String sentence = "Select CUSTID from LUIS.TRCUSTOMER";
for (int i = 0; i < 1; i++) {
    PreparedStatement ps = con.prepareStatement(sentence);
    ArrayList list = new ArrayList();
    ResultSet rs = ps.executeQuery();
    while (rs.next())
    {
        list.add(rs.getString("CUSTID"));
    }
    rs.close();
    ps.close();

    StringBuffer sb = new StringBuffer();
    for (Iterator iter = list.iterator(); iter.hasNext();) {
        sb.append((String) iter.next());
        sb.append(":");
        ...
    }
}
```

Avoiding the use of AutoCommit

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

Example 9-11 shows how to use your own transaction control.

Example 9-11 Using AutoCommit

```
public String doCommit() throws SQLException {
    EasyLog.writeLog(EasyLog.DEVELOPMENT, "Dbtest.doCommit()");

    PoolDB pool = new PoolDB("jdbc/myds");
    Connection con = null;
    PreparedStatement st = null;
    try {
        con = pool.connect();
        con.setAutoCommit(false);

        st = con.prepareStatement("update luis.employee set name = 'name_1' where emplno = '1'");
        int i = st.executeUpdate();
        st = con.prepareStatement("update luis.employee set name = 'name_2' where emplno = '2'");
        int j = st.executeUpdate();
        st = con.prepareStatement("update luis.employee set name = 'name_3' where emplno = '3'");
        int k = st.executeUpdate();

        con.commit();

    } catch (SQLException e) {
        con.rollback();
        e.printStackTrace();
    } catch (Exception e) {
        con.rollback();
        e.printStackTrace();
    }
    finally {
        if (st != null) st.close();
        if (con != null) con.close();
    }
    EasyLog.writeLog(EasyLog.DEVELOPMENT, "Dbtest.doCommit():end");
    return "done";
}
```

Important: Be careful. If you don't perform a commit and the connection is closed, a rollback is done automatically.

We run the code in Example 9-12 to show the difference.

Example 9-12 Setting AutoCommit test

```
// Using AutoCommit true
con = pool.connect();
con.setAutoCommit(true);

st = con.prepareStatement("...");
int i = st.executeUpdate();
st = con.prepareStatement("...");
int j = st.executeUpdate();
st = con.prepareStatement("...");
int k = st.executeUpdate();

con.commit();
```

```
// Using AutoCommit false
con = pool.connect();
con.setAutoCommit(false);

st = con.prepareStatement("...");
int i = st.executeUpdate();
st = con.prepareStatement("...");
int j= st.executeUpdate();
st = con.prepareStatement("...");
int k= st.executeUpdate();

con.commit();
```

We obtain the results in Table 9-5.

Table 9-5 AutoCommit test results

| Using... | Time to perform 200 iterations |
|-------------------|--------------------------------|
| AutoCommit(false) | 2.113 ms |
| AutoCommit(true) | 2.594 ms |

Data conversion

Database servers generally store data using a different encoding scheme than Java. The iSeries JVM stores string data as 2 byte Unicode. If the database is encoded in EBCDIC, the data is 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 `java.math.BigDecimal` class on every database read or write. You can avoid this conversion by storing numeric data in the database as float or double. Alternatively, consider converting the `BigDecimal` type to float for heavy calculations while leaving the database in decimal form.

Note: Rounding errors may be introduced by using float or double.

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 concept is designed purely to boost performance.

In the example in Figure 9-2, 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, turn off AutoCommit when doing batch updates.

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 9-2 Using batch insert

Isolation level

Isolation level represents the locking strategy when you access the database system. When an application runs, several clients can access the same data in the same period of time. This causes the following problems:

- ▶ **Dirty read:** Occurs when an application reads data from a database that has not been committed to permanent storage:
User A modifies a row. User B reads the same row before user A commits. User A performs a rollback. User B has read data that has never existed.
- ▶ **Nonrepeatable read:** Occurs when data has been changed between two consecutive reads of the same data:
User A reads a row but does not commit. User B modifies or deletes the same row and then commits. User A re-reads the row and finds it has changed (or it has been deleted).
- ▶ **Phantom read:** Occurs when a new set of data is inserted into a table between two read operations:
User A uses a search condition to read a set of rows but does not commit. User B inserts one or more rows that satisfy this search condition, then commits. User A re-reads the rows using the search condition and discovers rows that were not present before.

To solve these problems, you can apply one of the following isolation levels:

- ▶ **Read uncommitted:** This is the weakest isolation level. Although all of the problems can occur, this level gives better performance. You should not use it for important applications.
- ▶ **Read committed:** This level solves the dirty read problem. The data you read is always consistent. This level is useful for report-generating programs on report time.
- ▶ **Repeatable read:** This level solves dirty read and nonrepeatable read. It is useful when you have to update database records often. However, phantom reads may occur.
- ▶ **Serializable:** This is the strictest isolation level. This mode enforces all the ACID (Atomic, Consistent, Isolated, Durable) properties and guarantees fully independent transactions. It is useful for critical applications, although database access performance may suffer. This isolation level may create a lot of database locks, so use it only when it is necessary.

Table 9-6 shows the JDBC isolation levels, the correspondent isolation level in DB2, and whether it solves the problems.

Table 9-6 JDBC Isolation level to DB2

| Isolation level in JDBC | Isolation level in DB2 | Dirty read can occur | Nonrepeatable read can occur | Phantom read can occur |
|------------------------------|------------------------|----------------------|------------------------------|------------------------|
| TRANSACTION_SERIALIZABLE | Repeatable Read (RR) | No | No | No |
| TRANSACTION_REPEATABLE_READ | Read Stability (RS) | No | No | Yes |
| TRANSACTION_READ_COMMITTED | Cursor Stability (CS) | No | Yes | Yes |
| TRANSACTION_READ_UNCOMMITTED | Uncommitted Read (UR) | Yes | Yes | Yes |
| TRANSACTION_NONE | Not supported in DB2 | - | - | - |

The value *No* indicates that the problem does not occur. The value *Yes* indicates that the problem may occur.

The following example shows you how to change your connection isolation level:

```
PoolDB pool = new PoolDB("jdbc/myds");
Connection con = pool.connect();
con.setTransactionIsolation(Connection.TRANSACTION_READ_COMMITTED);
```

You can check the default isolation level on your connection by using:

```
con.getTransactionIsolation();
```

Table 9-7 shows the performance impact of different isolation levels.

Table 9-7 Isolation level performance impact

| Isolation level | Performance |
|------------------------------|-------------|
| TRANSACTION_READ_UNCOMMITTED | FASTEST |
| TRANSACTION_READ_COMMITTED | FAST |
| TRANSACTION_REPEATABLE_READ | MEDIUM |
| TRANSACTION_SERIALIZABLE | SLOW |

Scrollable result sets

The JDBC 1.0 API provides result sets that scroll in a forward direction only. JDBC 2.0 scrollable result sets allow for more flexibility in the processing of results by providing both forward and backward movement through their contents. In addition, scrollable result sets allow for relative and absolute positioning. An application can take advantage of this new feature to improve performance, especially when the result set is large.

You must be aware that scrollable result sets are slower than normal result sets.

Example 9-13 shows how to create a scrollable result set and how to iterate forward and backward.

Example 9-13 Using a scrollable result set

```
Statement st = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                   ResultSet.CONCUR_READ_ONLY);
ResultSet rs = st.executeQuery(sql);

StringBuffer sbf = new StringBuffer();
while (rs.next())
{
    sbf.append(rs.getRow());
    sbf.append("=");
    sbf.append(rs.getString(1));
    sbf.append(":");
}
EasyLog.writeLog(EasyLog.DEVELOPMENT,"Dbtest.doDbScrollable():forward "
                +sbf.toString());

StringBuffer sbb = new StringBuffer();
while (rs.previous())
{
    sbb.append(rs.getRow());
    sbb.append("=");
    sbb.append(rs.getString(1));
    sbb.append(":");
}
EasyLog.writeLog(EasyLog.DEVELOPMENT,"Dbtest.doDbScrollable():backward "
                +sbb.toString());
rs.absolute(1);
EasyLog.writeLog(EasyLog.DEVELOPMENT,"Dbtest.doDbScrollable():row="
                +rs.getRow()+"="+rs.getString(1));
rs.absolute(2);
EasyLog.writeLog(EasyLog.DEVELOPMENT,"Dbtest.doDbScrollable():absolute(2)-row="
                +rs.getRow()+"="+rs.getString(1));
rs.relative(-1);
EasyLog.writeLog(EasyLog.DEVELOPMENT,"Dbtest.doDbScrollable():relative(-1)-row="
                +rs.getRow()+"="+rs.getString(1));
```

The result for this code is:

```
Dbtest.doDbScrollable():forward==1=001      :2=002      :3=003      :4=004      :
Dbtest.doDbScrollable():backward=4=004      :3=003      :2=002      :1=001      :
Dbtest.doDbScrollable():row=1=001
Dbtest.doDbScrollable():absolute(2)-row=2=002
Dbtest.doDbScrollable():relative(-1)-row=1=001
```

If you are using absolute and relative fetch, do not use multiple fetch due to its random nature:

```
st.setFetchSize(1);
```

Tuning the block size

When you run an SQL statement for reading a database, several rows are retrieved from the table in a single request. In general, it is better to allow the database to determine the optimal blocking size. The default blocking size is 32 KB. In some cases, you can choose your own block size to tune the performance.

```
AS400JDBCConnectionPoolDataSource ds = new
AS400JDBCConnectionPoolDataSource();
ds.setBlockSize(64);
```

9.2.5 Caching your data source

In “Using the data source” on page 298, you learn how to use the data source PoolDB class. With this class, the datasource lookup code is encapsulated to your application, so you don’t need to code everywhere. However every time you ask this PoolDB class for a connection through the connect() method, the lookup is done. You can save a lot of time caching your datasource lookups.

Example 9-14 shows a class that caches all the datasource lookups for you.

Example 9-14 DataSourceFactory class

```
package com.ibm.rch.performance.java.db;

import java.util.HashMap;
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;

import com.ibm.rch.performance.log.EasyLog;

public class DataSourceFactory {

    private static DataSourceFactory singleton = null;
    private HashMap dataSourceCache = null;

    protected DataSourceFactory() {
        super();
        dataSourceCache = new HashMap();
    }

    public static DataSourceFactory getSingleton(){
        if (singleton == null)
            singleton = new DataSourceFactory();
        return singleton;
    }

    public DataSource getDataSource(String sdataSource) throws NamingException
    {
        DataSource ds = null;

        ds = (DataSource) dataSourceCache.get(sdataSource);
        if (ds != null)
        {
            EasyLog.writeLog(EasyLog.DEVELOPMENT,
                "DataSourceFactory.getDataSource():"+sdataSource + ":Found on Cache!!!");
        }else
        {
            EasyLog.writeLog(EasyLog.DEVELOPMENT,
                "DataSourceFactory.getDataSource():"+sdataSource + ":Not found. Let's lookup!!!");
            Hashtable parms = new Hashtable();
            parms.put(Context.INITIAL_CONTEXT_FACTORY,
                "com.ibm.websphere.naming.WsnInitialContextFactory");
            Context ctx = new InitialContext(parms);
            ds = (DataSource)ctx.lookup(sdataSource);
            dataSourceCache.put(sdataSource, ds);
        }
        return ds;
    }
}
```

```
}  
}
```

To obtain a data source and a connection from it, run this code:

```
DataSource ds = DataSourceFactory.getSingleton().getDataSource("jdbc/myds");  
Connection con = ds.getConnection();  
...
```

Then you can read the following information in the console:

```
DataSourceFactory.getDataSource():jdbc/myds:Not found. Let's lookup!!!  
DataSourceFactory.getDataSource():jdbc/myds:Found on Cache!!!  
DataSourceFactory.getDataSource():jdbc/myds:Found on Cache!!!  
...
```

9.3 Coding considerations with DDM (record-level access)

There is a significant difference between SQL access and record-level access. In SQL access, the programmer specifies what to retrieve. Then, the optimizer decides how to retrieve it. Where in the record-level access, 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 necessary record is read.

9.3.1 Minimizing open and close

Repeated execution of the `open()` and `close()` methods over the same file is expensive and unnecessary. Implement the program logic so that a file opens only once and remains open within the Java application.

9.3.2 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. Consider 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.

9.3.3 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 store the entire file.

9.4 Distributed Program Call

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 usually close 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 running unit, shutting down everything. However, the ILE COBOL/400® environment performs much better.

When using the Toolbox for DPC, you can only run batch programs on iSeries, and not interactive programs.

9.5 Coding consideration for servlets and JSPs

All the coding considerations mentioned previously also apply to coding servlets and JSPs. However, several coding conventions are specific to coding for a servlet engine. The following sections cover some of them.

9.5.1 Do not store large objects in HttpSession

Large applications require using persistent HttpSession. 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, an application forces WebSphere Application Server to process the entire HttpSession object each time. Also if the application passes the HttpSession remotely, the Application Server serializes and deserializes, but there is overhead.

When configuring persistent sessions in WebSphere Application Server, use a dedicated data source. To avoid contention for JDBC connections, don't reuse an application data source or the WebSphere Application Server repository for persistent session data.

As an alternative to storing the entire object 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.

9.5.2 Releasing HttpSession when finished

The application should explicitly and programmatically release 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), because it 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 invalidate method as shown in Example 9-15.

Example 9-15 Invalidating the HttpSession

```
public void logout(HttpServletRequest req, HttpServletResponse resp){
    // do Application specific operations to logout
    //...

    // invalidate the session
    HttpSession session = req.getSession(true);
    if (session != null){
        session.invalidate();
    }
}
```

9.5.3 Do not create HttpSession in JSPs by default

By default, JSP files create HttpSession. This is in compliance with Java 2 Platform, Enterprise Edition (J2EE), to facilitate the use of JSP implicit objects. This 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. To avoid the session from being created, insert the following tag in a JSP file:

```
<%@ page session="false"%>
```

9.5.4 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, avoid SingleThreadModel. Using SingleThreadModel does not prevent some synchronization problems such as static class variables.

If you want to be sure that you do not have synchronization problems, create a synchronized method and always call in the get method. This is not a good practice because this servlet does not scale. Be careful when using synchronized to define a method or a block.

```
public synchronized void dosync()
{
    // do some synchronized job
    ...
}
```

9.5.5 Using 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 is to cache any Java Naming and Directory Interface (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 9-4 on page 312.

9.5.6 Using the `HttpServlet destroy()` method

As you used the `init()` method for expensive operations and to cache some data as JNDI lookups, use the `destroy()` method to release these resources to avoid memory leaks.

9.5.7 Minimizing or eliminating the 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 WebSphere Development Studio Client, provide excellent debugging tools for unit testing. It also provides a WebSphere Test Environment where you can test your application as though you are using a real WebSphere Application.

However, even with these tools, there remains a legitimate need for application tracing both in test and production environments for error and debugging situations. Configure such application-level tracing as most system level traces to be activated in error and debugging situations only. We provide a sample Java log class in 9.1.6, “Logging” on page 291.

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. You achieve this by setting the file names to "" (two double quotation marks) in the Administrative Console. Be careful when using this feature because you do not have a log in case of bugs or exceptions. Use this method only to measure the amount of time expended on logging.

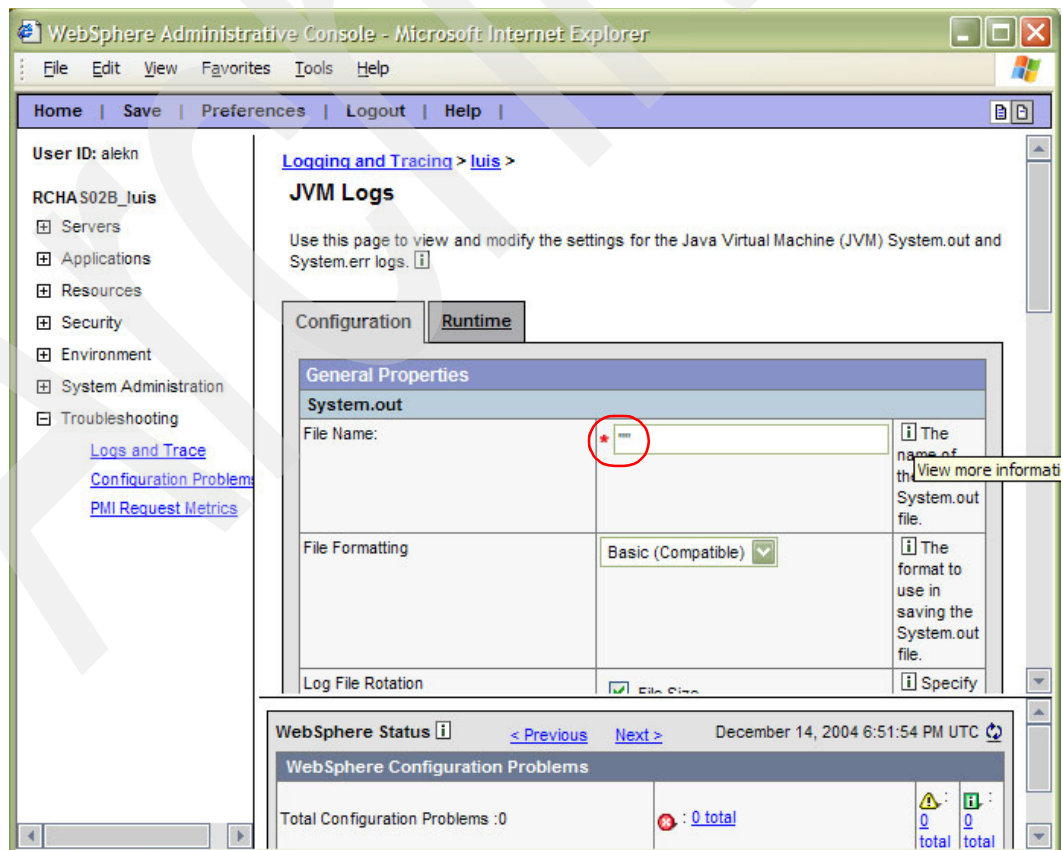


Figure 9-3 Setting JVM logs to ""

9.5.8 Don't use Beans.instantiate() to create new bean instances

The `java.beans.Beans.instantiate()` method 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.

9.5.9 Using and reusing data sources 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 9-4 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);
        Hashtable parms = new Hashtable();
        parms.put(Context.INITIAL_CONTEXT_FACTORY,
            "com.ibm.websphere.naming.WsnInitialContextFactory");
        Context ctx = new InitialContext(parms);
        // 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();
            ...
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } 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 9-4 Good JDBC coding technique

javax.sql.DataSource is obtained from WebSphere Application Server through a JNDI naming lookup. Avoid the overhead of acquiring javax.sql.DataSource for each SQL access. This is an expensive operation that severely impacts the performance and scalability of the application. Instead, servlets should acquire javax.sql.DataSource in the Servlet.init() method (or some other thread-safe method) and maintain it in a common location for reuse.

We run the code in Example 9-16 to show the difference.

Example 9-16 Caching the DataSource in the init() method

```
private javax.sql.DataSource ds = null;    // For Caching the DataSource

public void init() throws ServletException {

    super.init();
    EasyLog.writeLog(EasyLog.INFO,"TestServlet.init()");

    Hashtable parms = new Hashtable();
    parms.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");
    Context ctx;
    try {
        ctx = new InitialContext(parms);
        ds = (DataSource)ctx.lookup("jdbc/myds");
    } catch (NamingException e) {
        e.printStackTrace();
    }
    EasyLog.writeLog(EasyLog.INFO,"TestServlet.init():fin");
}

public String doget(long l, DataSource ds){

    for (int i = 0; i < l; i++) {
        Connection con = ds.getConnection();
        String sentence = "Select CUSTID from LUIS.TRCUSTOMER";
        PreparedStatement ps = con.prepareStatement(sentence);

        ResultSet rs = ps.executeQuery();

        rs.close();
        ps.close();
        con.close();
    }
    //catch...
}

public String doget(long l){

    for (int i = 0; i < l; i++) {
        PoolDB pool = new PoolDB("jdbc/myds");
        Connection con = pool.connect();
        String sentence = "Select CUSTID from LUIS.TRCUSTOMER";
        PreparedStatement ps = con.prepareStatement(sentence);

        ResultSet rs = ps.executeQuery();

        rs.close();
        ps.close();
        con.close();
    }
}
```

```
//catch...  
}
```

Then we obtain the results shown in Table 9-8.

Table 9-8 Caching the data source results

| Method | time to perform 100 iterations |
|-------------------------|--------------------------------|
| doget(long, DataSource) | 561 ms |
| doget(long) | 1102 ms |

9.5.10 Releasing 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 is reaped and returned by WebSphere Application Server after a time-out period, others may have to wait for this to occur.

Close the JDBC statements when you are finished with them. You can also explicitly close JDBC ResultSets. If not explicitly closed, ResultSets 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.

9.5.11 Page generation using strings

In servlets and JSPs, you use Java code to generate the HTML page to send the response to the user. Inside this code, you create some dynamic content as personalized tables with a certain schedule or different items to buy. In these cases, the code becomes an extremely large String. If you use a bad practice, this causes bad performance issues. Remember the coding techniques for Strings in 9.1.1, “String manipulation” on page 286.

In Example 9-17, we create a simple table using the correct coding technique.

Example 9-17 The createTableHTML method

```
public String createTableHTML(Collection col, String styleID, String styleDesc,  
                             String tableAtt)  
{  
    StringBuffer sb = new StringBuffer();  
    sb.append("<table ");  
    sb.append(tableAtt);  
    sb.append(">");  
    java.util.Iterator e = col.iterator();  
    while (e.hasNext())  
    {  
        ItemHTML row= (ItemHTML)e.next();  
        sb.append("<TR align=\"center\"> ");  
        sb.append("<TD style=\"");  
        sb.append(styleID);  
        sb.append("> ");  
        sb.append(row.getID());  
        sb.append("</TD>");  
        sb.append("<TD style=\"");  
        sb.append(styleDesc);  
        sb.append(">");  
        sb.append(row.getDescription());  
        sb.append("</TD></TR>");  
    }  
}
```

```

    }
    sb.append("</table>");
    return sb.toString();
}

```

9.5.12 Consideration with static content

Static content is something that can be served from an HTTP server instead of WebSphere Application Server. It is not worth the time to make WebSphere Application Server to do this job, so it can manage the dynamic content faster. This way the response time is shorter and the performance is better.

From a design point of view, we don't have to do anything so you can apply this technique to your old applications. For new applications, you may consider placing all your static content into a separate directory in your Web Content folder in the Web Application. This way the deployer sees quickly what to consider as static content and place it in the HTTP server documents folder.

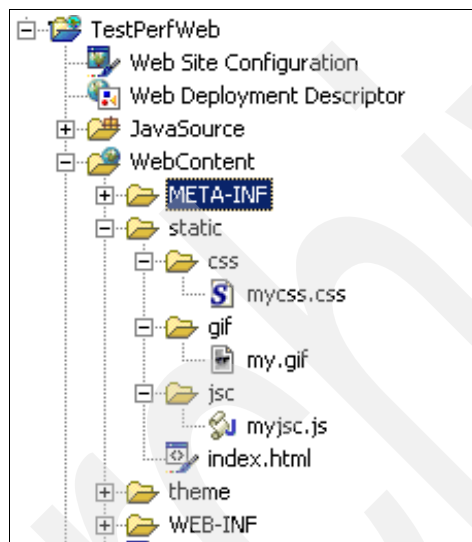


Figure 9-5 Placing static content into the Web application

See 6.6, “Static content location” on page 212, for more information.

9.5.13 JSP include directive versus Action

When you code a JSP, you may include other files into the JSP file, so part of the data presented to the user is written in another file. You can do this in two ways, which may impact the performance:

- ▶ If you use the include directive, you include a file into your JSP, which is resolved during compile time, so the performance is not impacted:

```
<%@ include file="myhtml.html" %>
```

- ▶ If you include an Action into your JSP, this is resolved during run time, so it impacts the performance. Review your code to make sure that you use an Action *only* where you need it.

```
<jsp:include page="myjsp.jsp"/>
```

9.5.14 Minimizing the useBean scope

When you create a bean, a bean scope is associated with it. The available scopes are:

- ▶ **page:** The bean is valid until the page sends the response to the user.
- ▶ **request:** The bean is valid from any JSP page that is processing the same request. You can use the request object to access the bean.
- ▶ **session:** You can use the bean from any JSP page in the same session.
- ▶ **application:** The bean is valid in any JSP in the same application.

Use the page scope unless otherwise necessary:

```
<jsp:useBean id="trainBean" scope="page"/>
```

9.6 Coding considerations for EJBs

Using EJBs with their own persistence mechanism provides a separate set of performance issues. This section covers some of the coding issues that relate to EJBs only.

9.6.1 Accessing 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. See Figure 9-6.

The WebSphere Application Server ships a complete command framework, which formalizes a solution to the problem presented in this section. The command facility is implemented in the `com.ibm.websphere.command` Java package.

You can learn more about the command framework on the Web at:

http://www-106.ibm.com/developerworks/websphere/registered/tutorials/0306_mcguinnes/mcguinnes.html

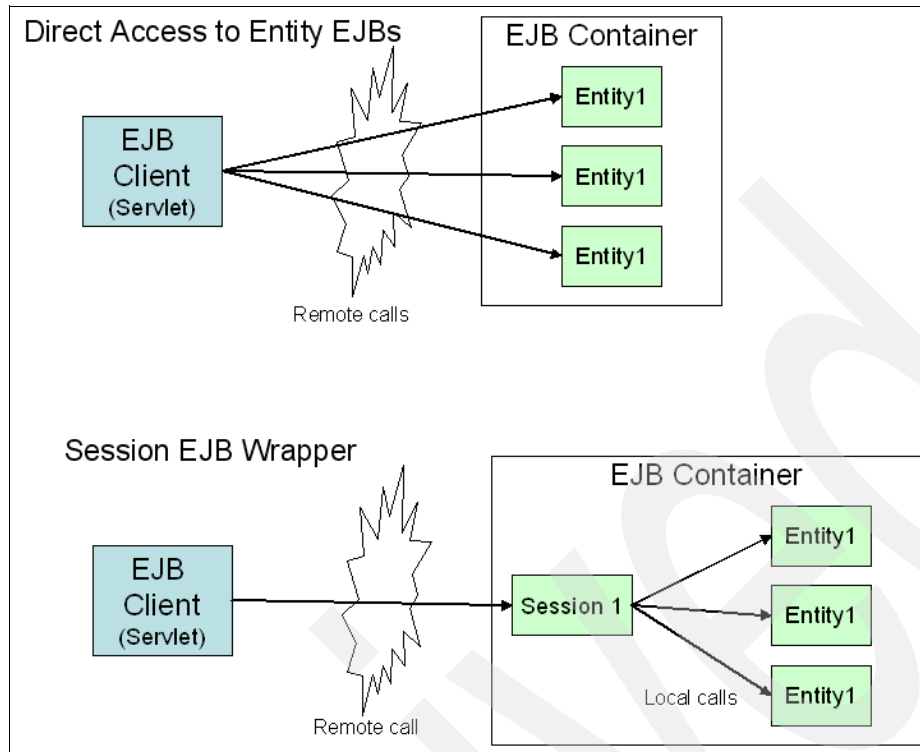


Figure 9-6 Session bean wrapper

9.6.2 JNDI names

The JNDI is the naming service that you use to locate objects, including EJBs, in WebSphere. The naming service maintains a set of bindings from the names to the EJB objects. When you create an EJB, you must create a JNDI name for it as shown in Figure 9-7.

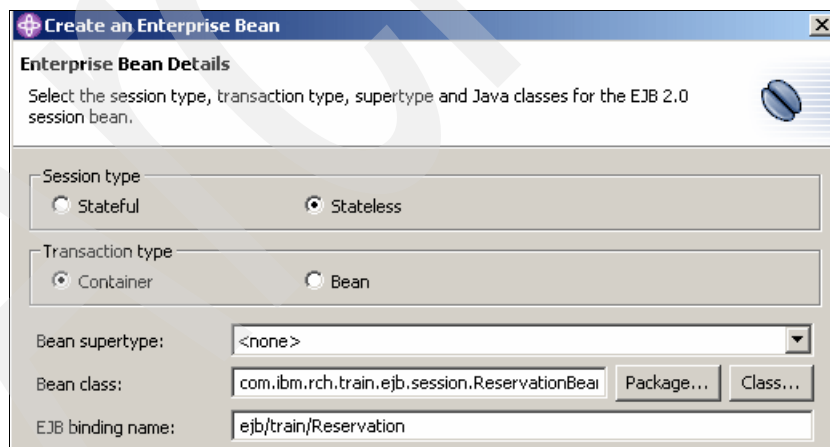


Figure 9-7 Creating a JNDI name for an EJB

The wizard gives you the default JNDI name for the EJB that you are creating. In default the JNDI name, `ejb/com/ibm/rch/train/ejb/cmp/ReservationLocalHome`, you receive all EJB packages and then the EJB name. This is a long JNDI name and your lookups will take a long time to reach the EJB.

The deeper a JNDI name tree is, the longer it takes to retrieve the names. Therefore, you can use a name convention and shorten the EJB JNDI name as in this example:

`ejb/train/Reservation`

To access an EJB from another EJB, you have to create a reference to it in the deployment descriptor. In doing so, you must follow the naming convention as shown in Figure 9-8. In the EJB Deployment Descriptor's References page, we create a reference to the reservation EJB. The reservation's EJB name is `ejb/train/Reservation` as shown in Figure 9-7. Now the Ticketing EJB can locate the reservation EJB.

The screenshot displays the 'References' page of the EJB Deployment Descriptor in WebSphere. On the left, a tree view shows the EJB structure: Customer, Purchase, Train, Reservation, and Ticketing. Under Ticketing, four EJBLocalRef entries are listed: ejb/train/Reservation, ejb/train/Train, ejb/train/Purchase, and ejb/train/Customer. The right pane shows the configuration for the selected reference. The 'Name' field is set to 'ejb/train/Reservation'. The 'Link' field is set to 'Reservation'. The 'Type' is set to 'Session'. The 'Local home' field is set to 'com.ibm.rch.train.ejb.session.ReservationLocalHome'. The 'Local' field is set to 'com.ibm.rch.train.ejb.session.ReservationLocal'. There are buttons for 'Browse...', 'Remove Link', and 'Add...'. The bottom pane shows 'WebSphere Bindings' with a JNDI name field containing 'ejb/train/Reservation'.

Figure 9-8 Creating an EJB reference

When reaching an EJB from the Web application, you have to create an EJB reference into the Web Deployment Descriptor, as shown in Figure 9-9. This reference can be local or remote. Use the same naming convention as in the previous example.

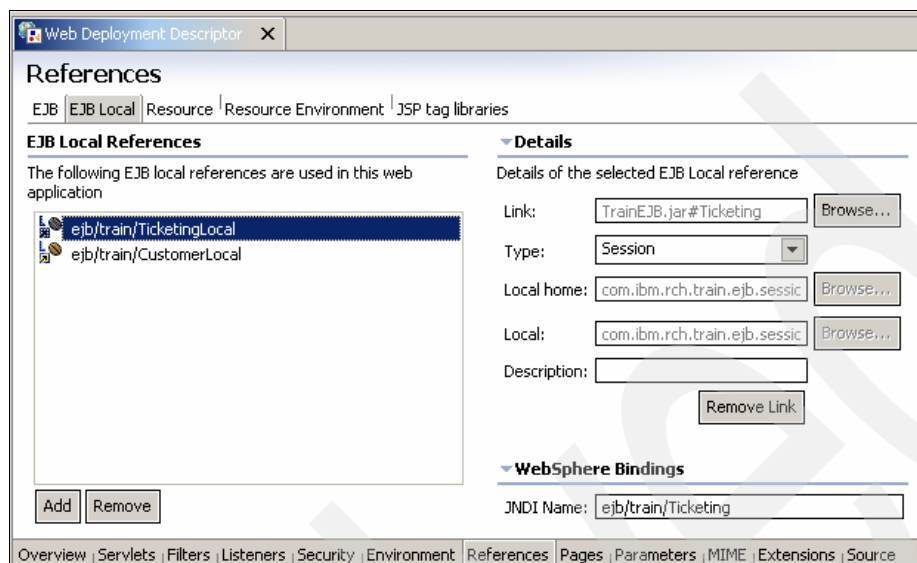


Figure 9-9 Defining EJB local references in the Web Deployment Descriptor

9.6.3 Reusing EJB home interfaces

EJB homes are obtained from WebSphere Application Server through a JNDI naming lookup. This is an expensive operation that you can minimize by caching and reusing EJB Home objects. For simple applications, it may be enough to acquire the EJB home in the servlet `init()` method (similar to the concept shown in Figure 9-4). 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. Example 9-18 shows one of these classes.

Example 9-18 *HomeFactory class*

```
import java.util.HashMap;

import javax.ejb.EJBHome;
import javax.ejb.EJBLocalHome;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import com.ibm.rch.performance.log.EasyLog;

public class HomeFactory {

    private static HomeFactory singleton = null;
    private static InitialContext initialContext = null;
    private HashMap homeCache = null;

    protected HomeFactory() {
        super();
        homeCache = new HashMap();
        try {
            initialContext = new InitialContext();
        } catch (NamingException namingException) {
```

```

        namingException.printStackTrace();
    }
}

public static HomeFactory getSingleton(){
    if (singleton == null) singleton = new HomeFactory();
    return singleton;
}

public Object getHome(String ejbRef) throws NamingException {
    if (initialContext != null){

        // Check for the home
        Object cacheObject = homeCache.get(ejbRef);
        if (cacheObject != null)
        {
            EasyLog.writeLog(EasyLog.DEVELOPMENT, "HomeFactory.getHome():"
                + ejbRef + ":Found on Cache!!");
            return cacheObject;
        }
        EasyLog.writeLog(EasyLog.DEVELOPMENT, "HomeFactory.getHome():"
            + ejbRef + ":Not found. Let's lookup!!");

        Object nsObject = initialContext.lookup
            (new StringBuffer("java:comp/env/").append(ejbRef).toString());

        if (nsObject instanceof EJBLocalHome){
            homeCache.put(ejbRef, nsObject);
            return nsObject;
        } else {
            EJBHome ejbHome = (EJBHome) javax.rmi.PortableRemoteObject.narrow
                ((org.omg.CORBA.Object)nsObject, EJBHome.class);
            homeCache.put(ejbRef, ejbHome);
            return ejbHome;
        }
    }
    else {
        throw new NamingException("HomeFactory: no InitialContext");
    }
}
}

```

Then as you run your application, you look up only the EJB the first time for each JNDI name. We run this code on Universal Test Client and see the console as shown in Figure 9-10.

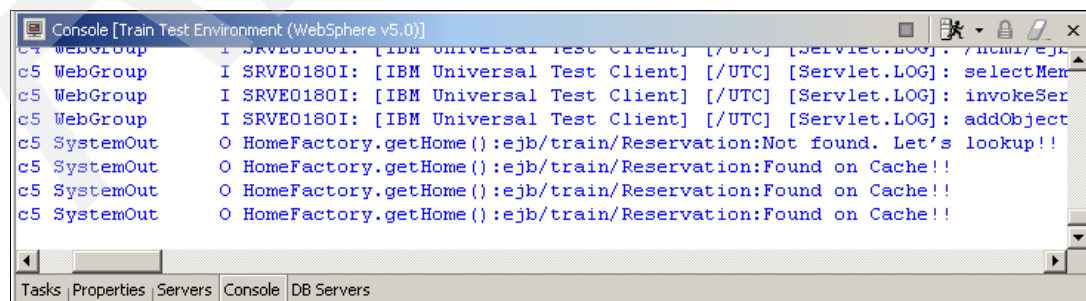


Figure 9-10 Console: Finding the JNDI name

We run a small program to show the difference (see Example 9-19).

Example 9-19 Caching the Home object test

```
//Caching using HomeFactory
    tiLocalHome =
        (TicketingLocalHome) HomeFactory.getSingleton().getHome(
            "ejb/train/TicketingLocal");
    tiLocal = tiLocalHome.create();
    String s = tiLocal.getLocal("Hello to everybody!!!");

// Not Caching
// InitialContext created before iteration
//InitialContext initialContext = new InitialContext();
    tiLocalHome =
        (TicketingLocalHome) initialContext.lookup(
            "java:comp/env/ejb/train/TicketingLocal");
    tiLocal = tiLocalHome.create();
    String s = tiLocal.getLocal("Hello to everybody!!!");
```

We obtain the results shown in Table 9-9.

Table 9-9 Caching the Home object test results

| Method | Time to perform 1.000 iterations |
|---------------------------|----------------------------------|
| Caching using HomeFactory | 80 ms |
| Not Caching | 370 ms |

9.6.4 Using info beans to reduce remote calls

When you perform a remote call, a lot of tasks must be performed including network traffic, and serializing and deserializing objects. Remote calls are expensive and decrease the performance. Use info beans to send the information response between the client and the EJB so you must decrease the number of the remote calls. For example, we create the class `CustomerInfo` to hold all the information of a single customer and then create a `getCustomerInfo` method to retrieve all the information at one time. See Example 9-20.

Example 9-20 Customer Info class

```
public class CustomerInfo {

    private String custID;
    private String firstName;
    private String lastName;
    private String password;
    private String address;

    public CustomerInfo(String _custID,
                        String _firstName,
                        String _lastName,
                        String _password,
                        String _address) {

        super();
        custID = _custID;
        firstName = _firstName;
        lastName = _lastName;
        password = _password;
        address = _address;
    }
}
```

```
/*
getters methods
...
*/
```

Now in the EJB client, we only call one method instead of five as shown in Example 9-21.

Example 9-21 Retrieving a CustomerInfo

```
CustomerInfo c = null;
try {
    CustomerHome customerHome = (CustomerHome ) HomeFactory
        .getSingleton().getHome("ejb/train/Customer");
    Customer customerEJB = customerHome.create(custID);
    // remote call
    c = customerEJB.getCustomerInfo();
} catch (NamingException e) {
    e.printStackTrace();
} catch (CreateException e) {
    e.printStackTrace();
} catch (RemoteException e) {
    e.printStackTrace();
}
```

We run the code in Example 9-22 to show the difference.

Example 9-22 The get bean test

```
public String dogetCustomerBean(long l){
    CustomerInfo c = null;
    for (int i = 0; i < l; i++) {
        try {
            CustomerLocalHome customerHome = (CustomerLocalHome ) HomeFactory
                .getSingleton().getHome("ejb/train/CustomerLocal");
            CustomerLocal customerEJB = customerHome.findByPrimaryKey("001");

            c = customerEJB.getCustomerInfo();

        } catch (NamingException e) {
            e.printStackTrace();
        } catch (FinderException e) {
            e.printStackTrace();
        }
    }
    return "done!";
}

public String dogetCustomer(long l){
    CustomerInfo c = null;
    for (int i = 0; i < l; i++) {
        try {
            CustomerLocalHome customerHome = (CustomerLocalHome ) HomeFactory
                .getSingleton().getHome("ejb/train/CustomerLocal");
            CustomerLocal customerEJB = customerHome.findByPrimaryKey("001");

            c = new CustomerInfo(
                (String)customerEJB.getPrimaryKey(),
                customerEJB.getFirstName(),
                customerEJB.getLastName(),
```

```

        customerEJB.getPassword(),
        customerEJB.getAddres());
    } catch (NamingException e) {
        e.printStackTrace();
    } catch (FinderException e) {
        e.printStackTrace();
    }
}
return "done!";
}

```

We obtain the results shown in Table 9-10.

Table 9-10 Get bean results test

| Method | Time to perform 40 iterations |
|---------------------|-------------------------------|
| dogetCustomerBean() | 1.051 ms |
| dogetCustomer() | 1.783 ms |

9.6.5 Releasing resources in EJBs

Following the EJB 2.0 specification, the container calls to the `ejbRemove()` method before removing a bean from the pool. If you acquire any resources in the bean, for example in the `ejbCreate()` method, release this resource in the `ejbRemove()` method. This applies to stateless session EJBs. For stateful session and entity EJBs, release the resources also in the `ejbPasivate()` method.

9.6.6 Using local interfaces when possible

In EJB 1.1, all clients had to use the remote interface even if they were in the same container. All calls implied network tasks and copying the objects to pass as parameters by value.

Using EJB 2.0, you can take advantage of the local interface. Then if the EJB client is located in the same JVM as your EJB, you can use the local interface to access the EJB. Then no network task is done and the parameters are passed by reference. This increases performance.

If you know that you will deploy your EJB clients in the same JVM where the EJB is deployed, you can use this local interface (see Figure 9-11).

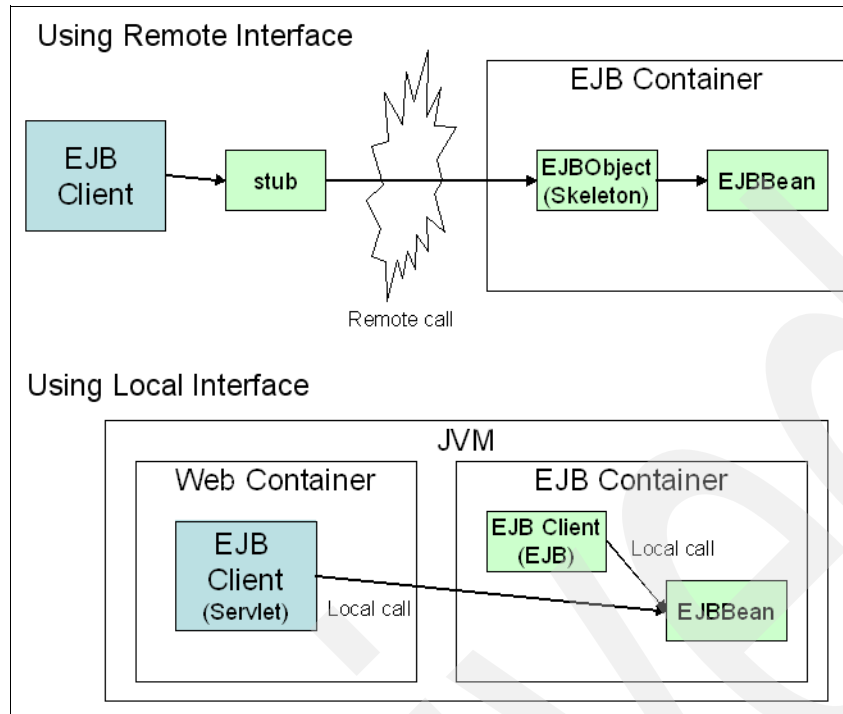


Figure 9-11 Local and the remote Interface

9.6.7 Removing stateful session beans when finished

Instances of stateful session beans have an affinity to a specific client. 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 may 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 passive session bean is subsequently required by the application, the container activates it by restoring it from disk. By explicitly removing stateful session beans when applications are finished using them, they decrease the need for making a bean passive and minimizing container overhead. To remove a stateful session bean, simply enter:

```
mySessionBean.remove();
```

9.6.8 Transactions

One of the most important reasons to choose EJBs to develop applications is because of their transaction capability. A *transaction* is the execution of a unit of work that accesses one or more shared resources. The ACID term defines what a transaction should be:

- ▶ **Atomic:** The transaction must be done completely or not be done at all.
- ▶ **Consistent:** After each transaction, the resources, mostly databases, must be consistent.
- ▶ **Isolated:** The data that a transaction accesses must be isolated from other transactions so it is blocked to others.
- ▶ **Durable:** After the transaction, the changes must be written somehow to a physical storage and endure in time.

Container managed transaction attributes

EJBs offer declarative transaction management. This way the developer doesn't need to handle all the resources that take place in the transaction. Instead, the container does this. You control the transactional behavior of EJBs through some transaction attributes in the deployment descriptor. You can define these attributes for each method of EJBs. You can set a transaction attribute for the entire EJB by using asterisk (*) as the method name.

The attributes are:

- ▶ **NotSupported:** The transaction scope is not propagated to this method.
- ▶ **Supports:** This method is included into the transaction scope if the transaction is started previously. It also accepts calls from a client with no transaction opened. Therefore, a transaction is not required to run this method.
- ▶ **Required:** This method must be executed in a transaction context. If a transaction is already opened, this method runs under it. However, if no transaction is opened, a new one is created by the container. This is the default behavior in a WebSphere environment.
- ▶ **RequiresNew:** A new transaction is always started when the method is called. It doesn't matter whether the method is called within a transaction.
- ▶ **Mandatory:** A transaction must be already running when the bean method is called. The method joins the transaction scope, but does not create a new one.
- ▶ **Never:** If the method is called in an existing transaction, then the container throws a `java.rmi.RemoteException`.

For methods that don't really have transaction requirements, apply the NotSupported attribute to them, so the container doesn't have to create the transaction for it.

Important: The EJB 2.0 specification advises that container-managed persistence (CMP) 2.0 entity beans use only the Required, RequiresNew and Mandatory transaction attributes. This is to ensure that all database activities are under a transaction scope. Support for Never, Supports, and NotSupported transaction attributes for CMP 2.0 is optional.

Isolation level for EJBs

EJB 2.0 has adopted the Java 2 Connector (J2C) resources isolation levels. Now you can specify the isolation by using the access intent in your EJBs. The isolation levels and the problems that they solve are discussed in "Isolation level" on page 304.

The access intent for each method in an EJB 2.0 CMP entity bean can be specified in the EJB deployment descriptor to provide hints to the EJB container about how the method will be used. The supported access intents are pessimistic update – Weakest Lock At Load, pessimistic update, optimistic update, pessimistic read, and optimistic read. In general, read-only methods and optimistic locking provide better performance. However, use care when using optimistic locking since problems may become apparent only under heavy load and may not be found in development.

Table 9-11 lists the access intent settings and how they may affect the underlying isolation levels.

Table 9-11 Access intent settings

| Access intent profile name | Concurrency control, access type | Transaction isolation |
|---|--|-----------------------|
| wsPessimisticRead | Pessimistic/Read (lock held for duration of transaction) | Repeatable Read |
| wsPessimisticUpdate | Pessimistic/Update (generates SELECT FOR UPDATE and grabs locks at beginning of transaction) | Repeatable Read |
| wsPessimisticUpdate-Exclusive | Pessimistic/Update (lock held for duration of transaction) | Serializable |
| wsPessimisticUpdate-noCollision | Pessimistic/Update (no locks are held, updates permitted, locks escalated on update) | Read Committed |
| wsPessimisticUpdate-WeakestLockAtLoad (default) | Pessimistic/Update (no locks are held, updates permitted, locks escalated on update) | Repeatable read |
| wsOptimisticRead | Optimistic/Read (read-only access) | Read Committed |
| wsOptimisticUpdate | Optimistic/Update (generates overqualified SQL UPDATE statements and forces compare before update) | Read Committed |

You must balance your performance needs against the data consistency. As mentioned earlier, when the isolation level is more restrictive, the performance become worse. Therefore, you have to study your application and see when you need one or another access intent. Some core entity beans can represent data involved in many transactions. If they don't have fast access, a bottleneck forms. However, because the data is used in many transactions, it must be consistent. You can apply different isolation levels to different methods.

The get methods are read-only methods, so you can apply a low isolation level as Read Committed. The set methods are more seldom used and need to be strongly isolated, so you can apply the most restrictive isolation level as Serializable.

Pessimistic methods block the data more often than optimistic methods, so normally optimistic methods are faster than pessimistic methods. See Figure 9-12.

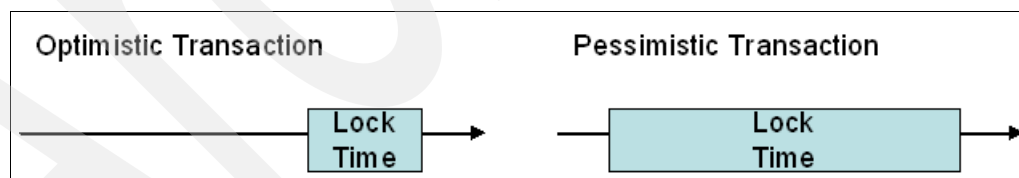


Figure 9-12 Lock time in optimistic and pessimistic transactions

You apply the access intent in the EJB Deployment Descriptor as shown in Figure 9-13.

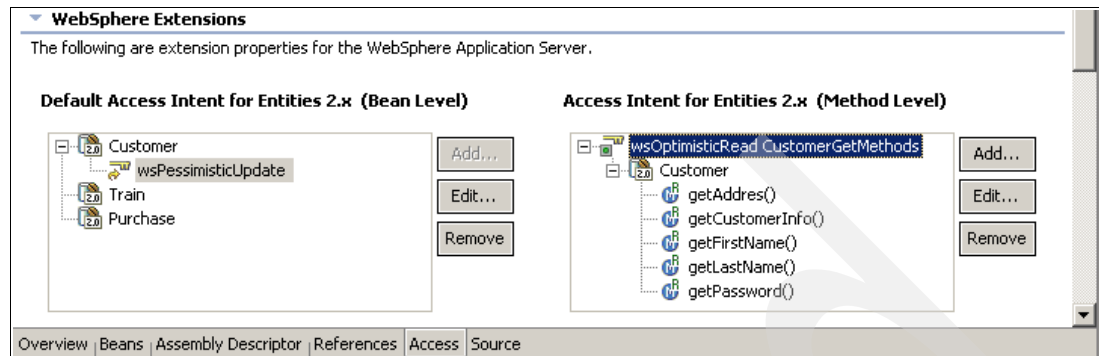


Figure 9-13 Defining the access intent for entity beans 2.0

Read-ahead hints

Read-ahead schemas enable applications to minimize the number of database round-trips by retrieving a working set of CMP beans for the transaction within one query. Read-ahead involves activating the requested CMP beans and caching the data of related beans (relationships). This ensures that data is present for the beans that are most likely to be needed next by an application.

A read-ahead hint is a canonical representation of the related beans that are to be read. It is associated with a finder method for the requested bean type, which must be an EJB 2.x compliant CMP entity bean.

9.6.9 Using CMPs instead of BMPs

The EJB 2.0 specification introduces such new features as EJB relationships and good performance improvements. Therefore, there is no need to develop the EJB as bean-managed persistence (BMPs). The container can monitor the bean's data in memory and only go to the database if any change occurs. The container can improve the performance in database operations, and you can take advantage of it using CMPs. Even with these performance improvements, in some cases good and accurate written BMPs can have better performance than CMPs.

9.6.10 Message-driven beans

Like session beans, message-driven beans (MDB) may also be modeled to represent tasks. However, they are invoked by the receipt of asynchronous messages. The bean either listens for or subscribes to messages that it is to receive.

From the performance point of view, there appears not to be any concern regarding MDBs. They are fast and can manage large amounts of messages without representing bottlenecks in the application.

9.6.11 Using home methods

EJB 2.0 specification introduces home methods for entity beans. These methods are not specific to an entity bean instance and must be implemented in the bean class with the signature `ejbHomeXXXX()`, for instance:

```
ejbHomegetTotalNumberOfCustomers()
```

When you call a home method, the container does not have to maintain the state data of a particular EJB so it performs better.

9.6.12 Using the `setEntityContext()` method

The `setEntityContext()` method is called only once for each bean. It is the first method called when the bean is created. Therefore, you can use this method to cache any data for the life of the bean.

Don't forget to use the `unsetEntityContext()` method to release any resource that you are holding in the bean. Otherwise you will have this resource held forever.

9.6.13 Don't use entity beans as simple data wrappers

When you create an entity bean, normally you think of it as an object to handle a row in a table, but this should not be the only purpose of an entity bean. They can hold business methods too. These business methods are close to where the data is. This increases performance avoiding remote calls.

9.6.14 Minimizing the use of stateful session beans

Minimize the use of stateful session beans or avoid it if possible. Unlike stateless session beans, they are unable to be workload managed or pooled. The EJB container may make inactive stateful session beans passive and then activate them if they are referenced later. This involves serializing and deserializing the beans, which can impact performance.

In cases where stateful session beans are required, you can remove occurrences of making beans passive by explicitly removing the beans when they are no longer required, instead of waiting for them to time out.

9.6.15 Using Data Class Access Beans

A Data Class Access Bean is an implementation of what is often referred to as *value object* or *data transfer object*. It is a container that holds a local copy of the selected entity bean attributes.

Each attribute has a getter and setter method. The Data Class keeps track of which attributes have changed by means of a *dirty flag*. When updating, only these dirty attributes are written back to the entity bean. You can significantly improve the performance with this feature.

Multiple Data Classes can be defined for an entity bean, each with a different set of cached properties. This can be useful to only retrieve those attributes that you are really going to use, especially if there are many of them.

When creating a data class, a factory is also created if the bean has a remote interface. This factory hides the lookup of the remote home interface and gives access to the create and finder methods.

There are three reasons for creating a Data Class:

- ▶ The EJB clients are simpler as they encapsulate all EJB methods.
- ▶ You can obtain a better performance because you manage EJB attributes locally through the bean instance.
- ▶ When you change any of the attributes, only the changed attributes are really updated in the database because of the dirty flags.

To create a Data Class for an EJB, follow these steps:

1. Open the J2EE perspective in WebSphere Development Studio Client.
2. Select the **EJB**, right-click, and select **New** → **Access Bean** as shown in Figure 9-14.

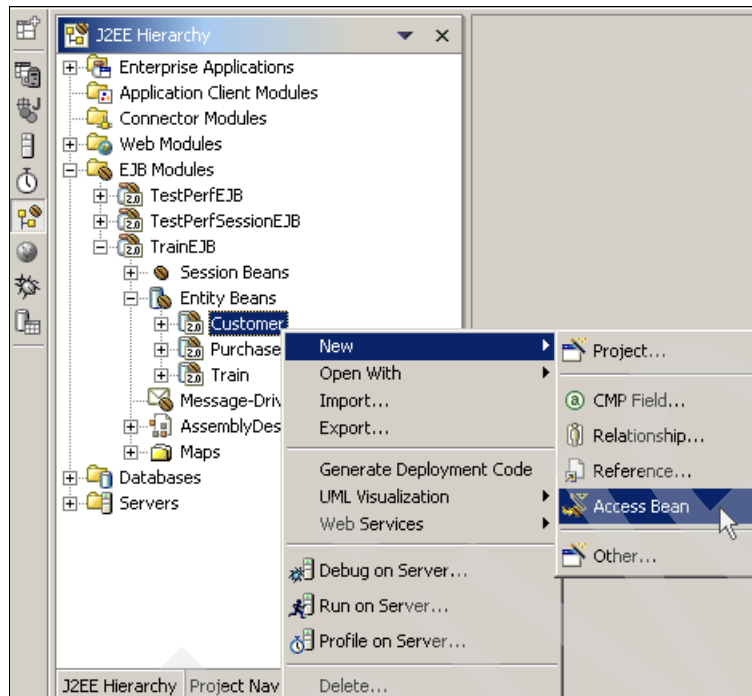


Figure 9-14 Creating a Data Class Access Bean

3. Select **Data Class** as the Access Bean type and choose the attributes you want to include into the Data Class. You can choose Local, Remote or both interfaces. You gain more benefits from creating Data Class for Remote Interface because of the decreasing number of getters and setters over the network, but also is applicable with the Local Interface.

As we created the Access Bean for our Customer EJB, the wizard creates a new class named CustomerData that is our new Data Class Access Bean.

Three new methods are created in the Customer EJB and promoted to the interface that you choose during creation the Data Class:

- ▶ `getCustomerData()`: This method obtains the CustomerData class with the EJB values.
- ▶ `setCustomerData(CustomerData)`: This method updates the EJB attributes with the new values from the CustomerData bean. If any attributes have changed in the EJB since the bean was obtained, a new `FieldChangedException` is thrown.
- ▶ `syncCustomerData(CustomerData)`: This method works the same as `setCustomerData`, but doesn't throw any exception.

Example 9-23 shows the code to manage the Data Class.

Example 9-23 Using Data Class Access Bean

```
String newLastName = "Aused";
String customerID = "001";
CustomerFactory customerFactory = new CustomerFactory();
Customer customer;
try {
    customer = customerFactory.findByPrimaryKey(custID);
    CustomerData customerData = customer.getCustomerData();
```

```

String firstName = customerData.getFirstName();
customerData.setLastName(newLastName);
customer.setCustomerData(customerData);
} catch (RemoteException e) {
    e.printStackTrace();
} catch (FinderException e) {
    e.printStackTrace();
} catch (FieldChangedException e) {
    e.printStackTrace();
}

```

9.7 Profiling your application

WebSphere Development Studio Client includes a profiling tool that you can use to study your application behavior during runtime. This tool gives you valuable information about your application and helps you to find the issues and bottlenecks. You can this tool to collect and display the data related to the runtime behavior of your application.

For the overview of this tool, see 4.5.1, “Profiling applications running on the WebSphere Application Server” on page 117. The following sections show several examples of using the Profiler.

9.7.1 Method speed

From the example in “Parsing numbers” on page 289, when you parse an int from a String, it’s better to use the Integer’s static method `parseInt(String)` instead of creating a new Integer object and call the `intValue()` method. With the Profiler, we prove this point by using these steps:

1. Start the server in profiling mode by clicking the button that is circled in Figure 9-15.

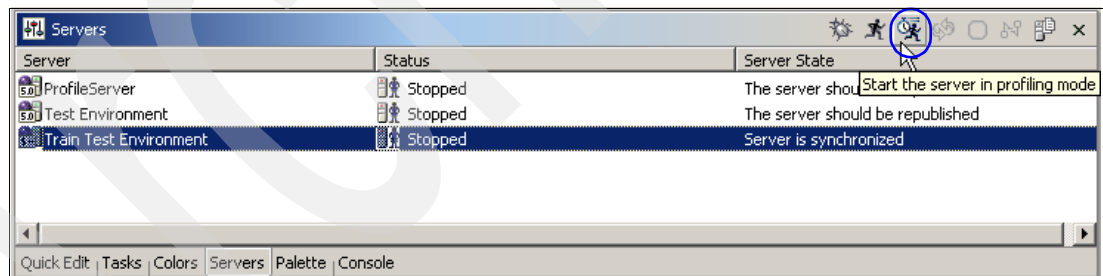


Figure 9-15 Starting the server in profiling mode

2. When the server starts, the Profile on server window (Figure 9-16) opens. Select **Java Profiling Agent** and click the > button to attach this agent to collect the data to profile your application. Click **Next**.

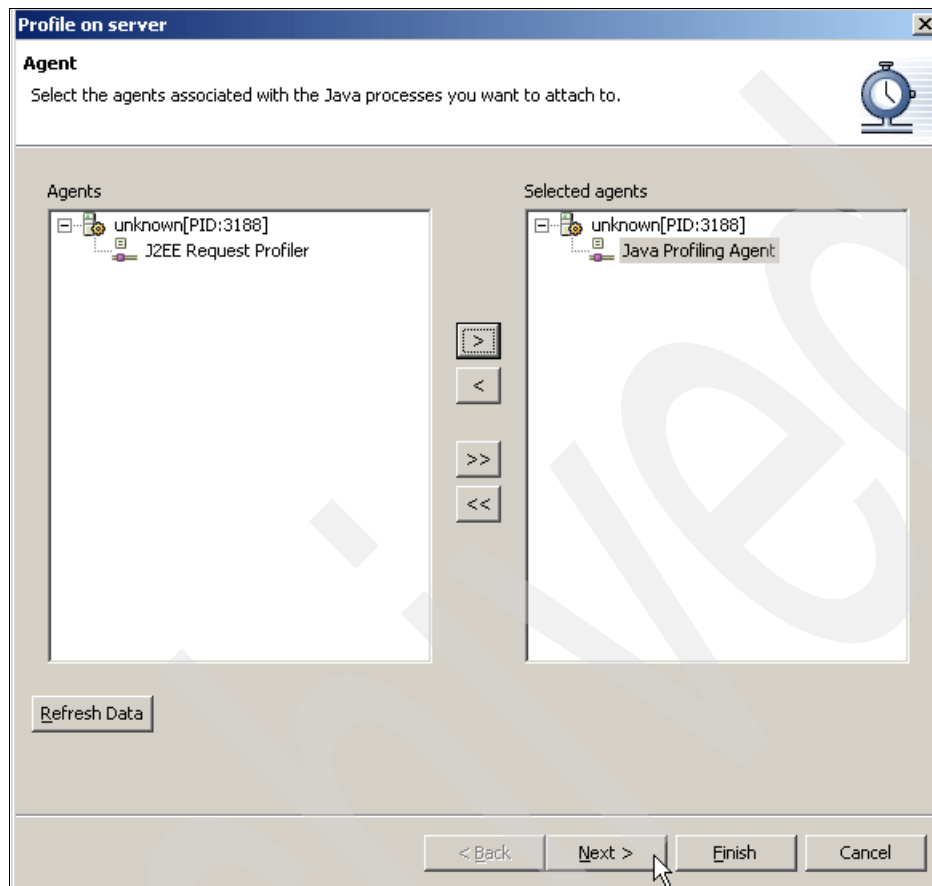


Figure 9-16 Profile on Server

3. In the next panel, leave the default values in Destination and click **Next**.

- In the Profiling Filters panel (Figure 9-17), select **WebSphere J2EE** as the filter set. We add `com.ibm.rch.*` to **INCLUDE** in the filter set. This is the common root of the packages we developed for the book. We want these packages to be included in the profiling example. We included `java.lang.String` class as well. The others filters are included in WebSphere J2EE filter set. Click **Next**.

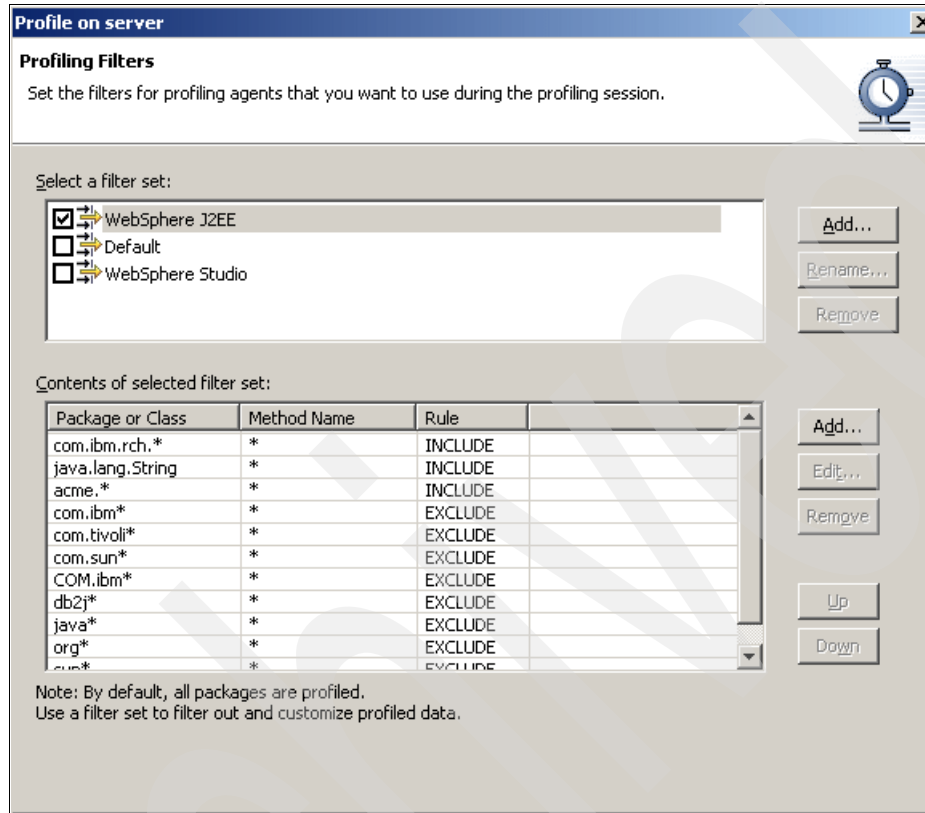


Figure 9-17 Defining Profiling Filters

- In the Profiling Options window (Figure 9-18), select both the **My application uses two much memory** and **My application is too slow** check boxes. Click **Finish**.

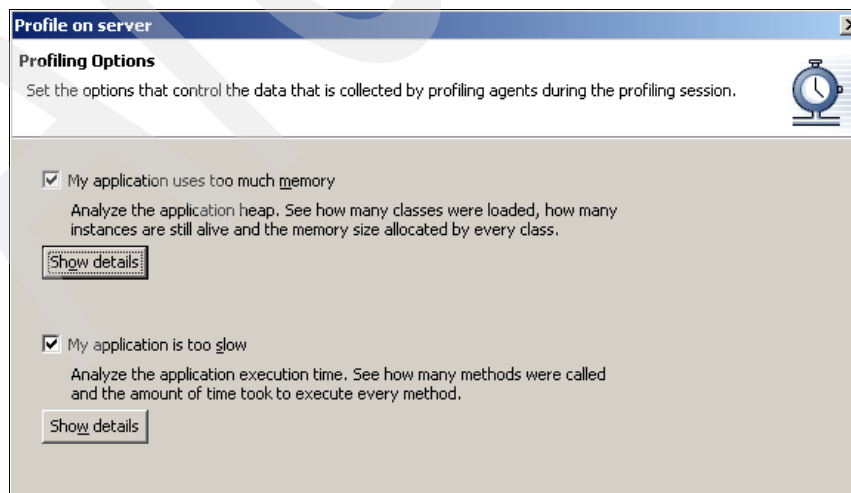


Figure 9-18 Profiling options

6. If the Profiling Tips window (Figure 9-19) opens, click **OK**.

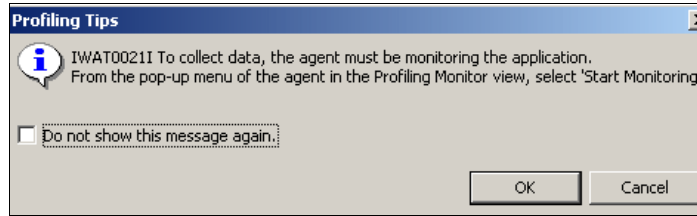


Figure 9-19 Profiling Tips

7. The Profiling and Logging Perspective opens in WebSphere Development Studio Client. We can see the server job in the Profiling Monitor view. The agent is attached to the server, but is not collecting data yet. The icon shows a *Paused* process. To start monitoring the server, select the **<Attached> Profiling** icon under the server, right-click, and select **Start Monitoring** as shown in Figure 9-20.

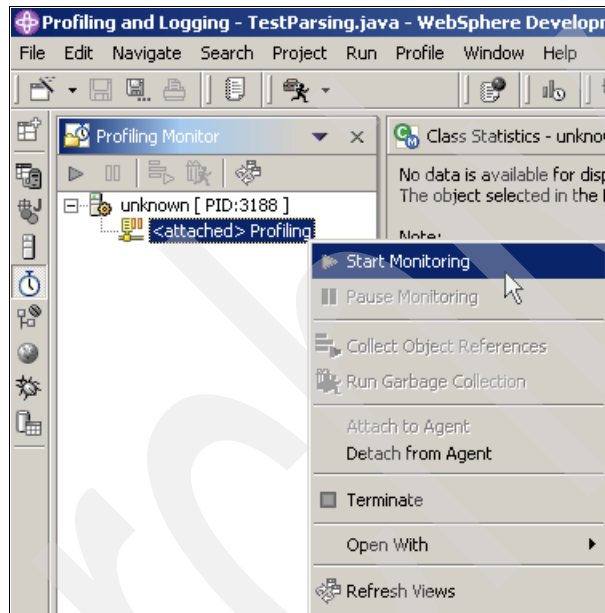


Figure 9-20 Starting the agent

A green play icon appears next to the attached agent as shown in Figure 9-21.

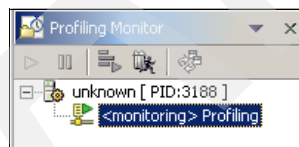



Figure 9-21 Play the attached agent

8. Now everything is ready to start seeing the monitored data. In a browser, access the page to make the first run on the server. After the first run, click the **Refresh views** () button.
9. Go to the Method Statistics view to see the results of this first test. This view allows you to view profiling data about particular methods. It is useful in identifying the time consuming methods in the application. It allows you to determine which methods are most frequently executed and which have the longest execution time.

There are several columns in this view, so click the **Class Names**'s title column to order the view by this column. Then find the TestParsing class since this is the class that holds the methods that you are trying to find. Remember that the objective is to know which method of this class is faster.

As you can see in Figure 9-22, TestParsing class is highlighted. Its first method is TestParsing(), which is its constructor.

| Method Names | <Class Names | Base Time | Average Base T... | Cumulative Time | Calls |
|---|-----------------|-----------|-------------------|-----------------|-------|
| getLastModifiedTime(java.io.File) l... | Win32FileSystem | 0.000237 | 0.000237 | 0.000237 | 1 |
| TestServlet() | TestServlet | 0.000200 | 0.000200 | 0.000200 | 1 |
| init() void | TestServlet | 3.056820 | 3.056820 | 3.156113 | 1 |
| doGet(javax.servlet.http.HttpServ... | TestServlet | 0.062817 | 0.062817 | 0.330103 | 1 |
| ejecutaOperacion(java.util.Propert... | TestServlet | 0.035659 | 0.035659 | 0.136001 | 1 |
| TestParsing() | TestParsing | 0.000008 | 0.000008 | 0.000008 | 1 |
| parseInt(java.lang.String, long) void | TestParsing | 0.000195 | 0.000195 | 0.000195 | 1 |
| parseIntStatic(java.lang.String, lon... | TestParsing | 0.000084 | 0.000084 | 0.000084 | 1 |
| length() int | String | 0.000302 | 0.000000 | 0.000302 | 1796 |
| indexOf(int) int | String | 0.741223 | 0.000323 | 0.741723 | 2296 |
| indexOf(int, int) int | String | 0.000801 | 0.000000 | 0.000801 | 3176 |
| equals(java.lang.Object) boolean | String | 0.000913 | 0.000000 | 0.000913 | 3380 |
| getChars(int, int, char[], int) void | String | 0.000103 | 0.000000 | 0.000103 | 282 |
| lastIndexOf(int, int) int | String | 0.000690 | 0.000000 | 0.000690 | 1765 |
| replace(char, char) java.lang.String | String | 0.003047 | 0.000001 | 0.003047 | 2477 |
| String(java.lang.String, java.lang.... | String | 0.012979 | 0.000001 | 0.012979 | 14294 |
| String(java.lang.StringBuffer) | String | 0.000215 | 0.000000 | 0.000215 | 756 |
| regionMatches(boolean, int, java.l... | String | 0.001377 | 0.000001 | 0.001377 | 1103 |
| startsWith(java.lang.String, int) b... | String | 0.000523 | 0.000000 | 0.000523 | 1329 |
| substring(int, int) java.lang.String | String | 0.000857 | 0.000000 | 0.000857 | 2223 |
| String(java.lang.String, java.lang.... | String | 0.001186 | 0.000002 | 0.001186 | 606 |
| indexOf(java.lang.String) int | String | 0.476287 | 0.000323 | 0.477566 | 1474 |
| indexOf(java.lang.String, int) int | String | 0.001279 | 0.000001 | 0.001279 | 1475 |
| charAt(int) char | String | 0.356624 | 0.000006 | 0.356624 | 58165 |
| hashCode() int | String | 0.000399 | 0.000000 | 0.000399 | 1730 |
| concat(java.lang.String) java.lang... | String | 0.000010 | 0.000001 | 0.000010 | 7 |
| equalsIgnoreCase(java.lang.String... | String | 0.000645 | 0.000003 | 0.000709 | 244 |
| endsWith(java.lang.String) boolean | String | 0.000015 | 0.000000 | 0.000015 | 52 |
| String(java.lang.String) | String | 0.000020 | 0.000001 | 0.000020 | 24 |

Figure 9-22 The Method Statistics view

Under this method, look for two methods. The columns you can look at are:

- **Base Time:** This is the total time spent running the method for the current row. It does not include time spent in other methods that are called by this method.
- **Average Base Time:** This is the average time spent in each execution of the method. In our example, this is the same as Base Time because we only run it once.
- **Cumulative Time:** This is the time spent executing the method, including any methods that are called by it. In our example, this is the same as Base Time because the method doesn't call any other methods.
- **Calls:** This is the number of times the method has been called, which in this example is only one.

As you can see we obtain 0.000195 as Base Time for the parseInt() method and 0.000084 as Base Time for the parseIntStatic() method. This make sense with the results shows in "Parsing numbers" on page 289, because we used almost twice as much time as when running the first method for the same number of iterations.

You can click the title of any column to order the view using this specific column as index. You can order the view according to the Base Time column to see the methods that spend more time in execution and look at this methods to see whether you can improve the performance.

10. After several calls, click **Refresh views** again. You now see that some yellow arrows appear on the increased measurements after the refresh as shown in Figure 9-23. This is helpful when you are looking for performance issues.

| Method Names | <Class Names | Base Time | Average Base Time | Cumulative Time | Calls |
|---|-----------------|-----------|-------------------|-----------------|-------|
| getLastModifiedTime(java.io.File) l... | Win32FileSystem | 0.000237 | 0.000237 | 0.000237 | 1 |
| TestServlet() | TestServlet | 0.000200 | 0.000200 | 0.000200 | 1 |
| init() void | TestServlet | 3.056820 | 3.056820 | 3.156113 | 1 |
| doGet(javax.servlet.http.HttpServ... | TestServlet | 0.098668 | 0.008222 | 12.037462 | 12 |
| ejecutaOperacion(java.util.Propert... | TestServlet | 0.038705 | 0.003225 | 11.710834 | 12 |
| TestParsing() | TestParsing | 0.000029 | 0.000002 | 0.000029 | 12 |
| parseInt(java.lang.String, long) void | TestParsing | 8.621599 | 0.287387 | 8.621599 | 30 |
| parseIntStatic(java.lang.String, lon... | TestParsing | 1.982172 | 0.066072 | 1.982172 | 30 |

Figure 9-23 Refreshing the view

When a new class becomes part of the view, a yellow rhombus is shown in the new class line circled in Figure 9-24.


| Method Names | Class Names |
|---------------------------------------|---|
| parseInt(java.lang.String, long) void |  TestParsing |

Figure 9-24 New class rhombus

Be careful that you don't mistake Base Time and Cumulative Time. Base Time is only the time spent in this method run. Cumulative Time adds the time spent in all the methods called by this method as shown in Figure 9-25.

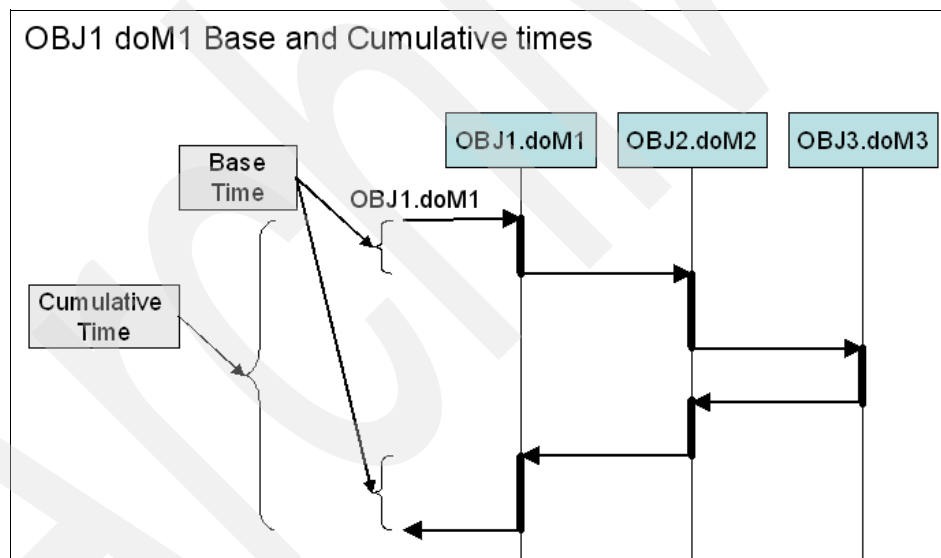


Figure 9-25 Base time and cumulative time

For example, order the view according to Cumulative Time as shown in Figure 9-26. The first method in the list is the Servlet method `doGet()`. The method with a bigger time to execute is `doGet`. This make sense because all the client calls start at this method and last until all other methods are done and the response for the client is ready to send back. Most of the work is done by other methods. If you look to Base Time, almost no time is spent in this method by itself.

| Method Names | Class Names | Base Time | Average Base Time | <Cumulative Time | Calls |
|--|-------------|-----------|-------------------|------------------|-------|
| <code>doGet(javax.servlet.http.HttpServlet...</code> | TestServlet | 0.053563 | 0.008927 | 4.633864 | 6 |
| <code>ejecutaOperacion(java.util.Propert...</code> | TestServlet | 0.026147 | 0.004358 | 4.443851 | 6 |
| <code>runTestParsing(long, int) java.lang...</code> | RunTest | 0.038696 | 0.012899 | 4.410009 | 3 |
| <code>parseInt(java.lang.String, long) void</code> | TestParsing | 2.642507 | 0.176167 | 2.642507 | 15 |
| <code>parseintStatic(java.lang.String, lon...</code> | TestParsing | 1.678869 | 0.111925 | 1.678869 | 15 |
| <code>init() void</code> | TestServlet | 0.128598 | 0.128598 | 0.130766 | 1 |
| <code>getParRequest(javax.servlet.http....</code> | Parametros | 0.094494 | 0.015749 | 0.094494 | 6 |
| <code>writelnLog(int, java.lang.String) void</code> | EasyLog | 0.084267 | 0.001139 | 0.084267 | 74 |
| <code>setEnd() void</code> | LogTime | 0.006648 | 0.000222 | 0.006715 | 30 |
| <code>setBegin() void</code> | LogTime | 0.004796 | 0.000160 | 0.004810 | 30 |

Figure 9-26 Cumulative time

For a definition of the columns, refer to “Measurement columns” on page 336.

Measurement columns

You can change the columns of the view. Right-click the view and select **Choose columns**. The Choose columns window (Figure 9-27) opens on which you can select the columns you want for the view.

Select the columns that you want displayed.
Use Up and Down buttons to change the order of the columns.

- ☒ Method Names
- ☒ Class Names
- ☐ Package
- ☒ Base Time
- ☒ Average Base Time
- ☒ Cumulative Time
- ☒ Calls

Up Down

Restore Defaults OK Cancel

Figure 9-27 Choose Columns for the view

For a better understanding of the measurement columns that you can display, read the column definitions in Table 9-12.

Table 9-12 Profiler column definitions

| Column name | Definition |
|-----------------|--|
| Total Instances | The total number of instances that were created for the selected package, class or method |
| Live Instances | The number of instances of the selected package, class, or method, where no garbage collection has taken place |

| Column name | Definition |
|---------------------------|---|
| Collected | The number of instances of the selected package, class, or method, that were removed during garbage collection |
| Total Size | The total size (in bytes) of the selected package, class, or method, of all instances that were created for it, including what was removed through garbage collection |
| Active Size | The sum size of all live instances |
| Base Time | For any invocation, the time taken to execute the invocation, <i>excluding</i> the time spent in other methods that were called during the invocation |
| Inherited Base Time | Similar to Base Time spent in the selected package or class, although it includes the time spent in other (inherited) methods that were called during the invocation |
| Cumulative Time | For any invocation, the time taken to execute the method including any methods that are called by it |
| Inherited Cumulative Time | Cumulative Time plus the time spent in inherited methods |
| Calls | The number of calls made by a selected method |
| Inherited Calls | The number of calls made by a method and by its inherited methods |

9.7.2 Finding memory leaks

A memory leak is a typical runtime problem when an object, that is not used by any part of the program, stays in memory for as long as the life span of JVM. In Java, you don't really manage memory resources. You instantiate objects. Then after using them, the object is no longer referenced and you trust garbage collector to free the memory used by unreferenced objects. Technically speaking, you cannot have memory leaks, but in fact, you sometimes have memory leaks.

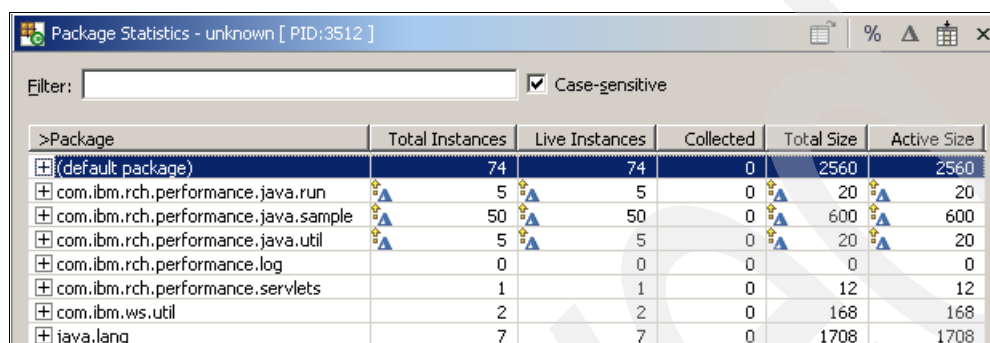
Garbage collector works on any unreferenced object. However, when you keep references to objects even if you think you don't, garbage collector cannot reach these objects. Then your application needs more and more memory resources as it runs. For more information, see Chapter 7, "Tuning the Java virtual machine" on page 217.

This section explains you to find memory leaks using Profiler.

1. Start the server the same way as in 9.7.1, "Method speed" on page 330.
2. Run the application under the load.

3. Refresh the views and go to the Package Statistics view. This view allows you to obtain a high-level view of which packages are most frequently used, and the memory used by objects instances of the classes in the packages. Because this is a general view, the numbers shown are bigger and easier to view. You see the first view of the application that is run as shown in Figure 9-28.

Since this is the first refresh, you can see the up arrows indicating that these measurements are increasing since the last refresh.




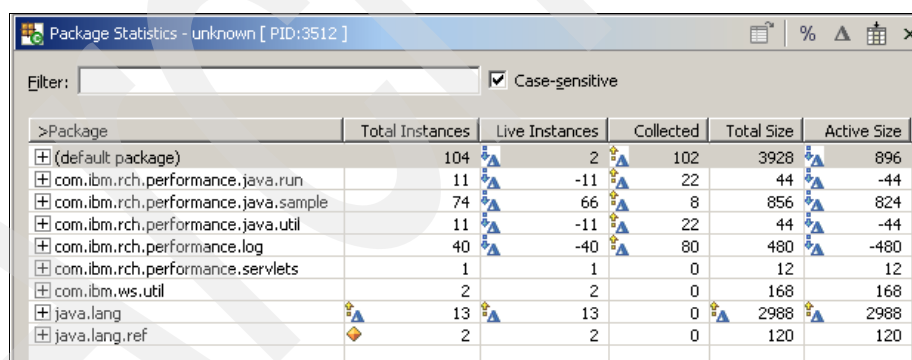
Package Statistics - unknown [PID:3512]

Filter: ☒ Case-sensitive

| >Package | Total Instances | Live Instances | Collected | Total Size | Active Size |
|-------------------------------------|-----------------|----------------|-----------|------------|-------------|
| (default package) | 74 | 74 | 0 | 2560 | 2560 |
| com.ibm.rch.performance.java.run | 5 | 5 | 0 | 20 | 20 |
| com.ibm.rch.performance.java.sample | 50 | 50 | 0 | 600 | 600 |
| com.ibm.rch.performance.java.util | 5 | 5 | 0 | 20 | 20 |
| com.ibm.rch.performance.log | 0 | 0 | 0 | 0 | 0 |
| com.ibm.rch.performance.servlets | 1 | 1 | 0 | 12 | 12 |
| com.ibm.ws.util | 2 | 2 | 0 | 168 | 168 |
| java.lang | 7 | 7 | 0 | 1708 | 1708 |

Figure 9-28 Profiling Package Statistics view

4. Run the application for some time and refresh the view again. The measurements continue to increase, which is normal behavior since no garbage collection is run yet.
5. Click the **Run Garbage Collection** button () to force garbage collector to run and free memory of not referenced objects. The garbage collector only runs when it is necessary to free memory, but since we are profiling, you can force it now.
6. You can see some down arrows indicating that the measurements are decreasing. This is the perfect application behavior, because the garbage collector found many objects and then freed the memory used by them. This view is shown in Figure 9-29.



Package Statistics - unknown [PID:3512]

Filter: ☒ Case-sensitive

| >Package | Total Instances | Live Instances | Collected | Total Size | Active Size |
|-------------------------------------|-----------------|----------------|-----------|------------|-------------|
| (default package) | 104 | 2 | 102 | 3928 | 896 |
| com.ibm.rch.performance.java.run | 11 | -11 | 22 | 44 | -44 |
| com.ibm.rch.performance.java.sample | 74 | 66 | 8 | 856 | 824 |
| com.ibm.rch.performance.java.util | 11 | -11 | 22 | 44 | -44 |
| com.ibm.rch.performance.log | 40 | -40 | 80 | 480 | -480 |
| com.ibm.rch.performance.servlets | 1 | 1 | 0 | 12 | 12 |
| com.ibm.ws.util | 2 | 2 | 0 | 168 | 168 |
| java.lang | 13 | 13 | 0 | 2988 | 2988 |
| java.lang.ref | 2 | 2 | 0 | 120 | 120 |

Figure 9-29 Decreasing measurements in the Profiler

You can see one column that has up arrows. This is the Collected column that indicates the number of objects collected since the last refresh. This column is increasing because of the garbage collector run. Again this is the normal behavior.

If you keep running this application, the number of objects continue increasing when refreshing. But after running garbage collector again, there are some lines that don't decrease the number of objects. See the `com.ibm.rch.performance.java.sample` line in Figure 9-30.

Package Statistics - unknown [PID:3512]

Filter: ☒ Case-sensitive

| >Package | Total Instances | Live Instances | Collected | Total Size | Active Size |
|-------------------------------------|-----------------|----------------|-----------|------------|-------------|
| (default package) | 147 | -27 | 174 | 7068 | -1644 |
| com.ibm.rch.performance.java.run | 31 | -31 | 62 | 124 | -124 |
| com.ibm.rch.performance.java.sample | 7987 | 7973 | 14 | 95788 | 95732 |
| com.ibm.rch.performance.java.util | 31 | -31 | 62 | 124 | -124 |
| com.ibm.rch.performance.log | 70 | -70 | 140 | 840 | -840 |
| com.ibm.rch.performance.servlets | 1 | 1 | 0 | 12 | 12 |
| com.ibm.ws.util | 3 | 3 | 0 | 252 | 252 |
| java.lang | 23 | 5 | 18 | 3772 | 2692 |
| java.lang.ref | 2 | 2 | 0 | 120 | 120 |

Figure 9-30 Profiler Package Statistics view

You can see that the Total Size is similar to the Active Size column. This means that most of the memory used by this package cannot be released. We click the + symbol to the left of the package name to drill down and see the classes included in it (see Figure 9-31).

Package Statistics - unknown [PID:3512]

Filter: ☒ Case-sensitive

| >Package | Total Instances | Live Instances | Collected | Total Size | Active Size |
|-------------------------------------|-----------------|----------------|-----------|------------|-------------|
| (default package) | 147 | -27 | 174 | 7068 | -1644 |
| com.ibm.rch.performance.java.run | 31 | -31 | 62 | 124 | -124 |
| com.ibm.rch.performance.java.sample | 7987 | 7973 | 14 | 95788 | 95732 |
| MemoryLeak | 7980 | 7980 | 0 | 95760 | 95760 |
| TestLoop | 5 | -5 | 10 | 20 | -20 |
| TestParsing | 2 | -2 | 4 | 8 | -8 |
| com.ibm.rch.performance.java.util | 31 | -31 | 62 | 124 | -124 |
| com.ibm.rch.performance.log | 70 | -70 | 140 | 840 | -840 |
| com.ibm.rch.performance.servlets | 1 | 1 | 0 | 12 | 12 |
| com.ibm.ws.util | 3 | 3 | 0 | 252 | 252 |
| java.lang | 23 | 5 | 18 | 3772 | 2692 |
| java.lang.ref | 2 | 2 | 0 | 120 | 120 |

Package Statistics | Class Statistics | Method Statistics | Object References | Instance Statistics

Figure 9-31 Show the classes included into the package

We found a class called MemoryLeak in this package. This class has the same number in Total Instances as in Live Instances, and the same Total Size as Active Size as well. It is clear that the garbage collector cannot handle these objects and the memory is leaked here. This was an easy case because we prepared this MemoryLeak class to show you how to find this issue, but it helped us to show you how to investigate the case if you have a similar problem in your application.

- Open the Class Statistics view. The data in this view provides the next lower level of details, after the Package Statistics. It shows which classes are most commonly used, the execution time for methods, and the size of the object. This makes it easier to identify particular classes that require further investigation. Click the **Total Size** column to sort the view as shown in Figure 9-32.

| Class Names | Package | Total Instances | Live Instances | Collected | <Total Size | Active Size |
|-------------------|--------------------------|-----------------|----------------|-----------|-------------|-------------|
| MemoryLeak | com.ibm.rch.performan... | 7980 | 7980 | 0 | 95760 | 95760 |
| [[char | (default package) | 147 | -27 | 174 | 7068 | -1644 |
| Class | java.lang | 13 | 13 | 0 | 3172 | 3172 |
| LogTime | com.ibm.rch.performan... | 70 | -70 | 140 | 840 | -840 |
| Thread | java.lang | 10 | -8 | 18 | 600 | -480 |
| ThreadPool\$... | com.ibm.ws.util | 3 | 3 | 0 | 252 | 252 |
| Parametros | com.ibm.rch.performan... | 31 | -31 | 62 | 124 | -124 |
| RunTest | com.ibm.rch.performan... | 31 | -31 | 62 | 124 | -124 |
| Reference\$... | java.lang.ref | 1 | 1 | 0 | 60 | 60 |
| Finalizer\$Fin... | java.lang.ref | 1 | 1 | 0 | 60 | 60 |
| TestLoop | com.ibm.rch.performan... | 5 | -5 | 10 | 20 | -20 |
| TestServlet | com.ibm.rch.performan... | 1 | 1 | 0 | 12 | 12 |
| TestParsing | com.ibm.rch.performan... | 2 | -2 | 4 | 8 | -8 |

Figure 9-32 Profile Class Statistics view

- The MemoryLeak class is in the first row (showing its bad behavior). Notice that the two most important facts when looking for memory leaks issues are that the Total Size column is quite similar to Active Size and the Live Instances is quite similar to Total Instances as well. This means that the garbage collector cannot reach a lot of these objects.

Click the **Show as Percentage** button (%).

You can see now that the Memory Leak class represent 88.58% of the Total Size and the 99.67% of the Active Size as shown in Figure 9-33. You can also notice that 0.00% objects have been collected.

| Class Names | Package | Total Instances | Live Instances | Collected | <Total Size | Active Size |
|----------------------------|-------------------|-----------------|----------------|-----------|-------------|-------------|
| MemoryLeak | com.ibm.rch.pe... | 96.20% | 96.20% | 0.00% | 88.58% | 99.67% |
| [[char | (default package) | 1.77% | -0.33% | 118.37% | 6.54% | -1.71% |
| Class | java.lang | 0.16% | 0.16% | 0.00% | 2.93% | 3.30% |
| LogTime | com.ibm.rch.pe... | 0.84% | -0.84% | 200.00% | 0.78% | -0.87% |
| Thread | java.lang | 0.12% | -0.10% | 180.00% | 0.56% | -0.50% |
| ThreadPool\$Worker | com.ibm.ws.util | 0.04% | 0.04% | 0.00% | 0.23% | 0.26% |
| Parametros | com.ibm.rch.pe... | 0.37% | -0.37% | 200.00% | 0.11% | -0.13% |
| RunTest | com.ibm.rch.pe... | 0.37% | -0.37% | 200.00% | 0.11% | -0.13% |
| Reference\$Reference... | java.lang.ref | 0.01% | 0.01% | 0.00% | 0.06% | 0.06% |
| Finalizer\$FinalizerThread | java.lang.ref | 0.01% | 0.01% | 0.00% | 0.06% | 0.06% |
| TestLoop | com.ibm.rch.pe... | 0.06% | -0.06% | 200.00% | 0.02% | -0.02% |
| TestServlet | com.ibm.rch.pe... | 0.01% | 0.01% | 0.00% | 0.01% | 0.01% |
| TestParsing | com.ibm.rch.pe... | 0.02% | -0.02% | 200.00% | 0.01% | -0.01% |

Figure 9-33 Class Statistics view in percentages

- The Profiler provides another very useful view to find memory leaks. This view is the Object References View. It displays the references from one object to another.

To collect the data to display in this view, in the Profiling Monitor view, select the attached **Profiling server**, right-click and select **Collect Objects References** as shown in Figure 9-34. This is necessary to take a dump of the heap.

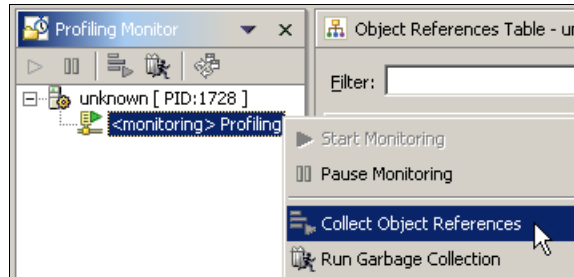


Figure 9-34 Collect Object References

- Refresh views and then open the Objects Reference Table as shown in Figure 9-35.

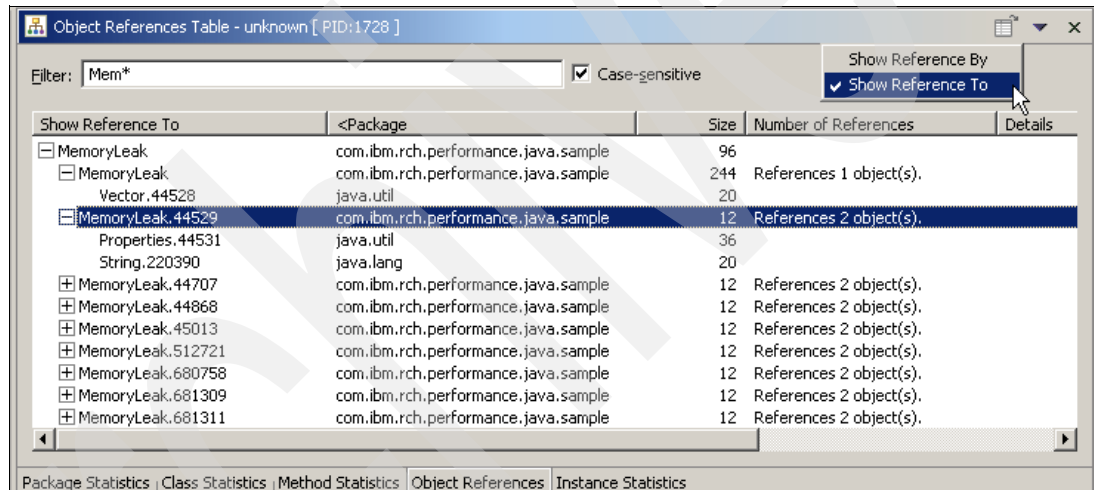


Figure 9-35 The Profiler Object References Table

By default, you see the instances that hold a reference to the selected instance. However, if you select **Show Reference To** (see Figure 9-35), then you see all references to other instances held by the selected instance. The Filter field can simplify the navigation in this view.

As you can see in Figure 9-35, MemoryLeak has references to other objects. The first line is for all the MemoryLeak objects. The second line is for the class variables referenced objects. A vector is referenced as a static variable. The third line is for the first MemoryLeak instance referenced. The number 44529 after the instance name is the object identification (OID), and means that is an instantiated object. This object references a Properties object and a String object.

For memory leaks, these references are not lost. If you run a garbage collection, all the references are still alive. As you run the application more and more, more references appear.

In a real life scenario, usually it is not so easy to find memory leaks. One reason is that the numbers are not always so clear in describing the problem. You need more time to investigate, and usually you don't name the problematic class *MemoryLeak*.

9.8 Using a response time watcher

After you tune your application, you deploy it to a server and then you have a well-tuned application in production. After some time, the application's response time may degrade and start growing because of a problem in the server. The number of customers could have grown or the system is shared by other applications more recently deployed. Regardless of the reason, the application may have some response time issues after some time.

In On Demand Business applications usually, it is difficult to know which part of the application is having the performance issue. A lot of different tiers are involved in On Demand Business applications, so a lot of different problems can happen. The only thing you know is that you have a bad overall response time. It takes a lot of time and effort to identify the problem, even if this is done by the people who developed the application.

You may consider providing the application with a system that alerts you whenever the application response time degrades. This system can be as complex as you want and can indicate the overall response time as well as where the issue is happening.

Example 9-24 show a simple way of doing this using the LogTime class.

Example 9-24 LogTime class to send an alert

```
public class LogTime {
    private long begin = 0;
    private long end = 0;

    public LogTime(){
    }

    public long getInterval() {
        return end - begin;
    }

    public void setEnd() {
        end = System.currentTimeMillis();
    }

    public void setBegin() {
        begin = System.currentTimeMillis();
    }

    public void watch(String process,
                      long maxTime){
        long interval = getInterval();
        if (interval > maxTime)
        {
            Timestamp t = new Timestamp(begin);
            EasyLog.writeLog(EasyLog.WARNING, "WatchTime:*****");
            EasyLog.writeLog(EasyLog.WARNING, "WatchTime: Process=" + process + "="+"started
at =" + t + "=");
            EasyLog.writeLog(EasyLog.WARNING, "WatchTime: Process=" + process + "="+"Lasted="
+ interval + "=");
            EasyLog.writeLog(EasyLog.WARNING, "WatchTime: maximum time was =" + maxTime + "=");
            EasyLog.writeLog(EasyLog.WARNING, "WatchTime: *****");
        }
    }
}
```

```
public String toString(){
    return "LogTime:=Interval (ms)="+getInterval()+"=";
}
}
```

The LogTime class helps to control which processes take more time in executing than normal. You see a warning in the system log whenever an execution takes a long time.

The LogTime class can be used for overall client response time putting it in the servlet's doGet() method or in any other place. Also you can add this watcher when you make a call to other systems to see if these systems exhibit the problematic behavior.

To use the LogTime class, you set the start and the end time of the process. Then you call the watch() method, including the name of the process that is running for further identification and the maximum time you think is reasonable to last. Example 9-25 shows sample code for using this LogTime class.

Example 9-25 LogTime class to set start and end time

```
public void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
    LogTime logTime = new LogTime();
    logTime.setBegin();

    // do
    //...

    logTime.setEnd();
    logTime.watch("TestServlet", 1000);
}
```

The alert system used in this class is the EasyLog class from 9.1.6, "Logging" on page 291. Eventually you can use other alert systems to send e-mail to a system administrator and to the developer responsible for this specific part of the application, so they can start fixing the problem as soon as possible.

Also you can keep these log timers in a database or in any other persistence system. This enables you to audit the response time that you had in the past. It is a good idea to print a daily report with the average response time for your system as well. You can build a more sophisticated version of LogTime with all these features.

Archived

Scaling and capacity planning for WebSphere applications

This chapter explores several aspects that you need to consider when you want to scale a WebSphere Application Server environment. It begins by summarizing the general factors that you need to explore if your business requirements demand a scalable solution. Then, it describes in greater detail some iSeries-specific characteristics that enable you to more easily scale your Java and WebSphere applications.

In addition, maintaining a highly scalable and reliable WebSphere Application Server infrastructure on the iSeries requires that you plan for adequate computing resources over time. With the proper planning, you can both maintain a stable environment and insure that you have the resources you need to manage changing business requirements. This chapter also examines a generic capacity planning process, which applies to all computing environments, and considers aspects that are iSeries-specific.

10.1 Scalability objectives

When considering scalability, your objective is to explore how a basic configuration can be extended to provide more computing power, by better exploiting the power of each machine or by using multiple machines. Specifically, you are interested in defining WebSphere Application Server configurations that exhibit the following properties:

- ▶ **Scalability:** The proposed configuration should allow the overall system to service a higher client load than that which is provided by the simple basic configuration. Ideally, it should be possible to service any given load, simply by adding the appropriate number of servers.
- ▶ **Load balancing:** The planned environment should ensure that each server in the configuration processes a balanced share of the overall client load that is being processed by the system as a whole. It would not do for one machine to be overloaded, while another machine is mostly idle. If all of the machines are of roughly the same power, each should process an equal share of the load. If various machines have different resource levels, each should process a portion of the load relative to its processing power.

Furthermore, if the total load changes over time, the system should naturally adapt itself to maintain this load-balancing properly, regardless of the actual total magnitude of this load. All machines may be used at 50% of their capacity, or all machines may be used at 100% of their capacity. You generally want to begin capacity planning when overall system usage reaches 60% to 70% utilization. Refer to 10.6, “Capacity planning and workload estimation tools” on page 362, for more information about capacity planning.

- ▶ **Failover:** The idea of having multiple servers naturally leads to the potential for the configuration to provide automated failover. That is, if any one server were to fail for any reason, the configuration should continue to operate with the remaining servers. The load-balancing property should ensure that the client load is redistributed to the remaining servers. Each of the remaining servers then process a proportionately slightly higher percentage of the total load. Of course, such an arrangement presupposes that the system is designed with some degree of over capacity, so that the total number of servers are indeed sufficient to process the total expected client load.

Ideally, the failover aspect should be totally transparent to users of the configuration. 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, some failover solutions may not be completely transparent.

For example, a client that is currently in the middle of an operation, when a server fails, may receive an error from that operation and may be required to retry (at which point the client is 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 using a different server.

The important point for failover is that each client, and the set of clients as a whole, can continue to take advantage of the system and receive useful service, even if some of the servers fail or become unavailable. Conversely, when a previously failed server is repaired and again becomes available, the system should transparently start using that server again to process a portion of the total client load.

The failover aspect of a configuration is also sometimes called *fault tolerance*, in that it allows the system to survive a variety of scheduled or unscheduled failures or faults. Note, however, that failover is only one technique in the much broader field of fault tolerance. No such technique can make a system 100% safe against any possible failure. The goal is to greatly minimize the probability of a 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 Application Server also allows one server process on a given machine to fail independently, while other processes on that same machine continue to operate normally.

Aside from these primary attributes, also consider the number of secondary characteristics when evaluating a configuration with scalability in mind:

- ▶ **Dynamic changes:** 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 are highly desirable, since they enhance the overall manageability and flexibility of the system.
- ▶ **Mixed configuration:** In some scenarios, it may be possible to mix multiple versions of a server or application, 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 affect upgrades without any interruption of service.
- ▶ **Fault isolation:** In the simplest application of failover, you are concerned only 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. However, there are sometimes situations where one malfunctioning server may, in turn, create problems for the operation of other, otherwise healthy, servers.

For example, one malfunctioning server may hoard system resources, or database resources. Or 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:** Distributing processing tasks among multiple servers and, in particular, multiple machines, 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.

10.2 Scaling a solution

This section looks at various aspects of scalability. It starts with workload patterns and continues by looking at the steps that are relevant to scaling your application.

10.2.1 Workload patterns

Most high-volume Web sites experience traffic patterns that can be characterized as “bursty”. This refers to patterns with 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 enterprises do business across multiple time zones around the globe, the burstiness factor may level out somewhat, 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**

Publish/subscribe sites include search engines, media sites such as newspapers and magazines, and event sites such as those for the Olympics and for 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 likely has little or no connection to any legacy systems.

► **Online shopping sites let users browse and buy**

Shopping 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 retailers range from less than one million hits per day to over 3 million hits per day or more. Of the total transactions, typically between 1% and 5% are buy transactions. When users make a purchase, security requirements become significant and include privacy, integrity, authentication, and current regulatory requirements.

Shopping sites may 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**

Examples of self-service sites include banking from home, tracking packages, and making travel arrangements. Data comes largely from legacy applications and often comes from multiple sources, thereby potentially 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, while 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), and the most complex transactions. They 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 generally 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 potential 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.

10.2.2 Scalability in practice

Meeting the challenges provided by the various types of Web sites requires flexibility and capacity for rapid 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 that 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 multi-tiered enterprise solution from end to end means managing the performance and capacities of each component within each tier.

Figure 10-1 shows a basic topology with the typical and major components. From a scalability viewpoint, the key components of an enterprise solution are:

- ▶ Web server
- ▶ Application server
- ▶ Network
- ▶ Directory and security servers
- ▶ Firewalls
- ▶ Business servers
- ▶ Database server

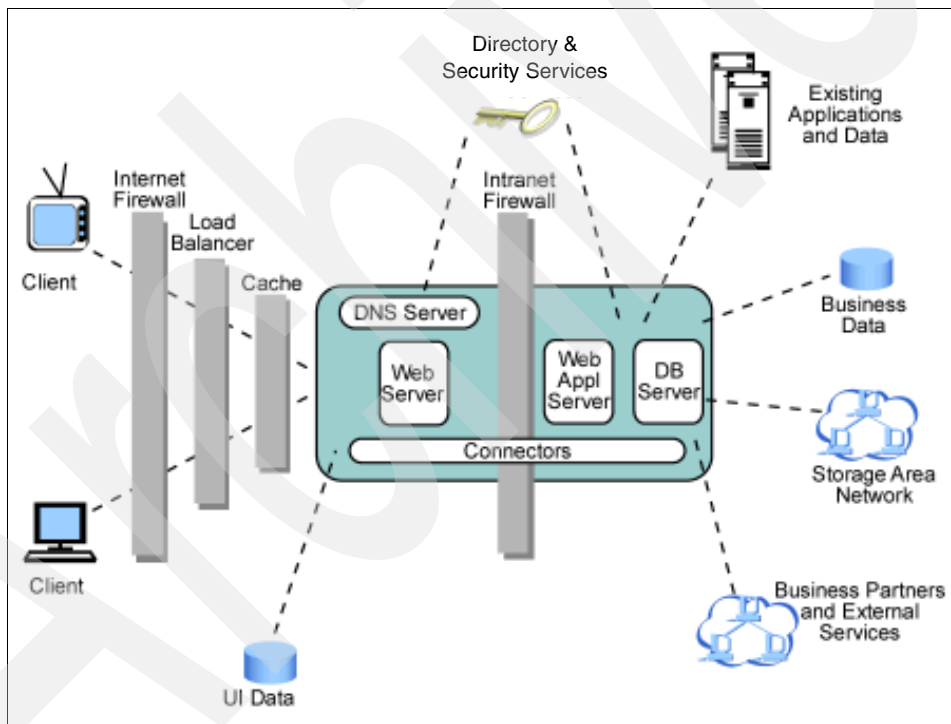


Figure 10-1 Scaling applications: A simple Web site topology

The core objectives in scaling a component or system are:

- ▶ Increase capacity or performance
- ▶ Improve efficiency and reliability
- ▶ Shift or reduce the load

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 10-1 shows how the three principle scaling objectives may be met in terms of the various scaling techniques.

Table 10-1 How scaling techniques relate to scaling objectives

| | Scaling techniques | Increase capacity or performance | Improve efficiency | Shift or reduce load |
|---|-------------------------|----------------------------------|--------------------|----------------------|
| 1 | Use a faster machine | X | | |
| 2 | Create server clusters | 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 |

10.2.3 Six steps to scaling your solution

This section describes a high-level, systematic approach to classify your Web site (applications) so you can determine which scaling techniques to apply. You need to improvise and adapt the approach depending on the specifics of your situation.

The six steps involve:

1. Understanding your environment
2. Categorizing your workload
3. Determining the components most impacted
4. Selecting the scaling technique or techniques to apply
5. Applying the technique or techniques
6. Evaluating or re-evaluating the outcome

Step 1: Understanding your environment

For existing configurations, 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 your 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 from the beginning. Each technique affects application design. Likewise, application design impacts the effectiveness of the technique. To achieve proper scale, application design must consider the effects of potential scaling. In the absence of known workload patterns, you'll need to follow an iterative, incremental approach.

Step 2: Categorizing your workload

Your site may have high-volumes of dynamic transactions. The type of site you have (see 10.2.1, “Workload patterns” on page 347) will become clear as you evaluate your site for the other characteristics that pertain to transaction complexity, volume swings, data volatility, security, and so on.

Step 3: Determining the components most affected

This step involves mapping the most important site characteristics to each component. Again, from a scalability viewpoint, the key components of an enterprise solution are the Web server, the Web application server, the network, the directory and security servers, the firewalls, the business servers, and the database servers.

Step 4: Selecting the scaling techniques to apply

It is worth your best efforts to collect the information that is needed to make the best scaling decision. Only when the information gathering is as complete as possible, it is time to consider matching scaling techniques to components. Here's a summary of the eight scaling techniques:

► Using 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 issue is that software capabilities can limit the hardware exploitation and vice versa. Another issue is that, due to hardware or software changes, you may need to also change existing system management policies.

► Creating a cluster of machines

This method 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 your deployment. The IBM WebSphere Application Server uses an efficient data sharing technique to support clustering.

► Using 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 fast and serve a specific role.

Examples of these are network appliances and routers with cache, such as the IBM 2216. Some issues to consider regarding special machines are the sufficiency and stability of the functions. Such issues also includes the potential benefits in relation to the added complexity and manageability challenges. Caching may reduce the HTTP server traffic significantly.

► Segmenting the workload

This method applies to the Web server, the Web application server, the network, the intranet firewall and the database. The goal is to split 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.

To implement segmentation, you need to characterize the different workloads serviced by the component. After segmenting the workload, additional infrastructure is required to balance the physical workload across the segments. For example, you may need to implement the WebSphere Application Server's Network Deployment (ND) version.

► **Batching 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 requestors and responders by allowing the requestor 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 requestor due to the elimination of the overhead costs with multiple requests. Challenges with this method may include the limited ability to reuse requests due to inherent differences in various requests types. Supporting different request types can lead to increased costs.

► **Aggregating user data**

This method applies to the Web server, the Web application server, and the network. The goal allows rapid access to large customer data controlled by existing system applications and to support personalization based on customer-specific data. When accessing customer data spread across existing system applications, the applications may be overloaded, especially when the access is frequent. This can degrade response time. To alleviate this problem, this 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 large number of customers. A CIS *must* scale very well to support large data as well as field requests from a large number of requesters.

► **Managing 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 and to eliminate the overhead involved in 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 connection managers that support 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. Within the WebSphere Application Server, the connection manager supports *connection pooling*. The key issue with this technique is maintaining a session's identity when several sessions share a connection.

► **Caching**

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 thereby reduce 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. When these changed pages are generated, they must be effectively cached for subsequent data requests. Intelligent caching technologies can significantly enhance the scalability of enterprise systems.

The key issues with caching dynamic Web pages are determining what pages to cache and when a cached page has become obsolete.

Step 5: Applying the technique or techniques

Apply the selected technique or techniques in a test environment to evaluate the performance or scalability impact to the component, as well as the impact to its surrounding components. You certainly do not want a situation where improvements in one component result in an increased and unnoticed load on another component.

Factors that may 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: Evaluate and re-evaluate

As with all performance-related work, tuning is required. The goals are to eliminate bottlenecks, scale to a manageable status those bottlenecks that can't be eliminated, and work around those bottlenecks that can't be scaled.

Some of the tuning actions that are used and the benefits that may be realized are listed here:

- ▶ Increasing the Web server threads raises the number of requests that can be processed in parallel.
- ▶ Adding indexes to the database server reduces input/output (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.

While scaling end-to-end enterprise solutions remains more of an art than a science, this approach provides a systematic method to respond to and manage unpredictable demands on your Web site. With a thorough knowledge of your current workload patterns, Web site components, skills, and budget, you should now begin to factor scalability considerations and techniques into future planning.

10.3 iSeries-specific scaling considerations

The iSeries server has several significant characteristics that can assist you in scaling a WebSphere Application Server environment. Of greatest significance are the implementations of the Java virtual machine (JVM) and the database (DB2 Universal Database (UDB) for iSeries), both of which are integrated into the operating system.

The iSeries implements the concept of a Technology Independent Machine Interface (TIMI). This allows the JVM and database to be integrated tightly into the operating system System Licensed Internal Code (SLIC) to provide a highly tuned, efficient, stable, and scalable implementation. Figure 10-2 illustrates how TIMI is implemented on the iSeries.

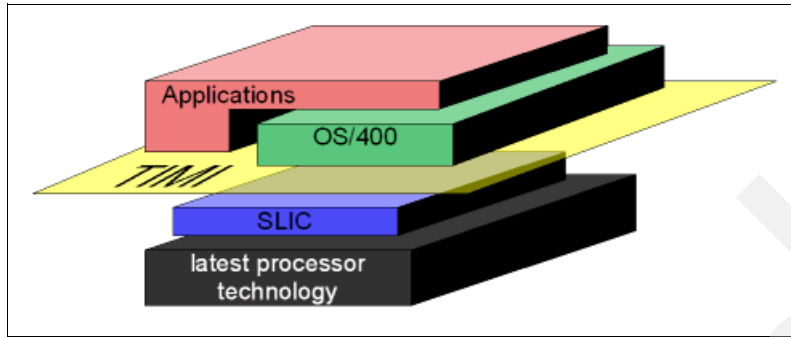


Figure 10-2 Scaling applications: TIMI

10.3.1 iSeries advantages and configuration customization options

The iSeries offers many different advantages over other platforms for running WebSphere Application Server-based applications, including ease of operations and administration. The total cost of ownership (TCO) is proven lower on the iSeries. As a multi-user and multiapplication platform, you won't necessarily need separate machines to run each part of your Web application.

Specific iSeries advantages include:

- ▶ Integrated DB2 database
- ▶ Integrated JVM
- ▶ Access to legacy applications
- ▶ Multiple instance support

Configuration customization options include:

- ▶ Knowing when the server startup has completed
- ▶ Accessing administrative functions via a Web browser
- ▶ Executing administrative functions via script
- ▶ Creating WebSphere clusters (vertical, horizontal)
- ▶ Configuring HTTP servers
- ▶ Using automated deployment
- ▶ Implementing within a heterogeneous environment
- ▶ Using Java Database Connectivity (JDBC) drivers

10.3.2 WebSphere for iSeries scalability overview and key concepts

This section provides a conceptual overview of the goals and issues associated with scaling applications in the WebSphere Application Server for iSeries environment.

Vertical scaling considerations

Vertical scaling provides a straightforward mechanism for creating multiple instances of an application server, and therefore, multiple JVM processes. In the simplest case, you can create many application servers on a single machine, and this single machine also runs the HTTP server process.

We recommend that you plan vertical scaling configurations ahead of time. However, since they do not require any special installation steps, you can always implement them later on an as-needed basis. Although there is a limit to the resources available on a single machine, and an availability risk when using a single machine, this configuration provides process isolation.

It may also provide improved throughput over a configuration where only a single application server process is running.

This assumes, of course, that sufficient central processing unit (CPU) and memory are available on the machine (and that the application can take advantage of such a configuration). Therefore, when implementing a vertically scaled WebSphere environment, make sure that you do not exceed the available resources.

Important: The best way to ensure good performance in a vertical scaling configuration is to tune a single instance of an application server for throughput and performance, then incrementally add additional instances. Test performance and throughput as each application server instance is added. When CPU utilization reaches 85% or greater, it's likely that adding additional instances will do little to improve performance and throughput as the operating system starts performing context switching between processes. Always monitor memory use when you are configuring a vertical scaling topology. Do not exceed the available physical memory on a machine.

Some reasons to use vertical scaling are:

► **Performance:** Better use of the CPU

An instance of an application server runs in a single JVM process. However, the inherent concurrency limitations of an application in the JVM process may prevent it from fully using the processing power of a machine. Creating additional JVM processes provides multiple thread pools, each corresponding to the JVM associated with each application server process. This can enable the application in the application server or servers to make the best possible use of the processing power and throughput of the host machine.

► **Availability:** Failover support in a cluster

A vertical scaling topology also provides process isolation and failover support within an application server cluster. If one application server instance goes offline, the requests to be processed are redirected to other instances on the machine.

► **Throughput:** WebSphere workload management

Vertical scaling topologies can use the WebSphere Application Server workload management facility. The HTTP server plug-in distributes requests to the Web containers and the Object Request Broker (ORB) distributes requests to Enterprise JavaBean (EJB) containers.

► **Maintainability:** Easy to administer member application servers

With the concept of cells and clusters in WebSphere Application Server, it is easy to administer multiple application servers from a single point.

► **Scalability:** Vertical scalability can easily be combined with other topologies

You can implement vertical scaling on more than one machine in the configuration (IBM WebSphere Application Server Network Deployment V5.1 must be installed on each machine). You can combine vertical scaling with other topologies to boost performance and throughput. Again, this assumes, of course, that sufficient CPU and memory are available on the machine.

► **Cost:** Does not require additional machines

Horizontal scaling considerations

Horizontal scaling exists when the members of an application server cluster are located across multiple physical machines. This lets a single WebSphere application span several machines, yet still present a single logical image. Figure 10-3 shows one such topology that

you may consider. In this example, the application server layer is clustered. You may also consider clustering at the HTTP and database layers for greater performance gains.

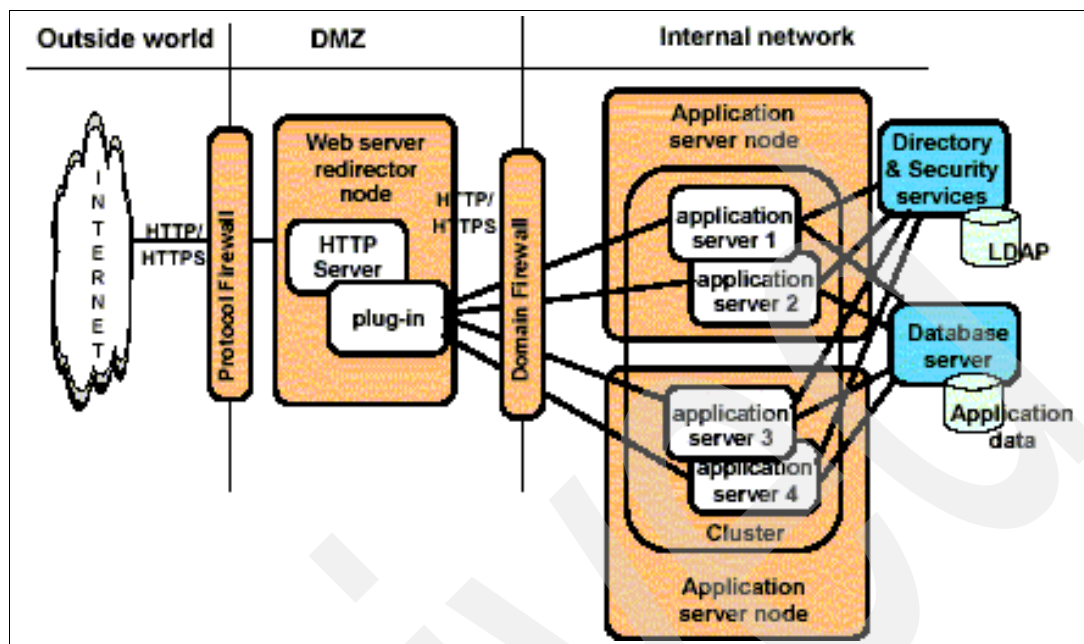


Figure 10-3 Scaling applications: Using horizontal clustering

Advantages

Horizontal scaling using clusters has the following advantages:

► Availability

It provides the increased throughput of vertical scaling topologies but also provides failover support. This topology allows handling of application server process failures and hardware failures without significant interruption to client service.

► Throughput

It optimizes the distribution of client requests through mechanisms such as workload management or remote HTTP transport.

Disadvantages

Horizontal scaling using clusters has the following disadvantages:

► Maintainability

With the concept of cells and clusters in IBM WebSphere Application Server Network Deployment V5.1, it is easy to administer multiple application servers from a single point. However, more installation and maintenance are associated with the additional machines.

► Cost

You need to invest in additional infrastructure, such as in more machines.

You can find additional information about vertical and horizontal scaling in *IBM WebSphere V5.1 Performance, Scalability, and High Availability WebSphere Handbook Series*, SG24-6198.

Other considerations

Here are some other points to consider:

- ▶ When you multiple instances of the WebSphere Application Server are configured on a single iSeries server, ensure that the port information specified on one instance does not conflict with the configuration specified on the other instances. You can use the WebSphere Application Server `dspwasinst` script from Qshell to view the port configuration. You may also want to use the `-portblock` parameter when creating instances so you can manage the ports better and prevent conflicts.
- ▶ If you plan to use persistent HTTP sessions in your environment and use horizontal scaling techniques for WebSphere workload management, create a persistent data source or JDBC driver using the IBM Toolbox for Java JDBC driver.
- ▶ If you are only using vertical scaling techniques, then use the iSeries Native JDBC driver.

EJB workload management

A Java client issues a method call on an EJB remote reference. In WebSphere applications, you can enable the EJB stubs using Workload Manager (WLM). This process causes the stub to become aware of the existence of replicas of the enterprise beans that may exist on separate application servers.

From the client perspective, there is no difference in calling a method on a bean that is enabled by Workload Manager or a normal bean. The WLM-enabled stubs dynamically route the method call to any one of the available instances of the bean. There are 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.

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, but contain pure "logic".

Entity beans can be cloned too, because each of the clones can retrieve the state from the RDB. However, 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 recommend that you keep servlets and EJBs in the same JVM that 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, such as a shopping cart. The use of stateful session beans is hazardous, since it can lead to disastrous performance in some cases. We recommend that you maintain the context as attributes of the HTTP session, by using persistent HTTP sessions and stateless session beans. Attempt to minimize amount of data stored in the HTTP session. You must use persistent sessions so they can be shared by multiple instances. The architecture is more fault tolerant, but still remains client-specific.

In this case, scalability is ensured by servlet clustering. The HTTP requests are distributed across a number of available servers. 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 on the same server is good practice in this scenario, because it reduces Internet InterORB Protocol (IIOP) 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 is still accessed on Server A, with a strong performance premium being paid.

10.3.3 JDBC driver choices on the iSeries

You have a choice of two different JDBC drivers to consider when accessing data on the iSeries. The choices are the IBM Toolbox for Java JDBC driver and the Native JDBC driver.

When running applications on the iSeries server using WebSphere Application Server, use the native JDBC driver whenever your database exists on the same server. It performs better. When your WebSphere Application Server exists on one iSeries server and your database is on another iSeries server, use the Toolbox driver.

10.4 Planning future capacity to maximize performance

How much time and effort 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 over time, too. Managing your system, at first glance, may seem like just another time-consuming job. But the investment quickly pays off because the system runs more efficiently. 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. However, there are times when those resources are not in the right place. Maybe additional systems and clients have been added to the network, or production volume has increased and, as a result, the workload is significantly changed. Or, more often, the workload changes in small, nearly invisible increments, and one day, performance is not as good anymore. That is why you have to plan ahead for your system to be at peak performance.

It is important to manage performance effectively for both today and tomorrow. For the short term, understanding the performance components of your system helps you react quickly when a performance problem occurs. 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. Moreover, your users receive the excellent service they expect. Maintaining or exceeding acceptable service levels 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 the strategy of planning, implementing, controlling, and measuring computer-based tasks to achieve performance that is, at a minimum, acceptable. For example, performance never exceeds the maximum acceptable response time indicated in a service level agreement.

You should also be aware that some tuning results do not happen immediately after a change. There is a delay or lag before the effect is seen. This is indicated in Figure 10-4, which shows a composite cause-effect graph.

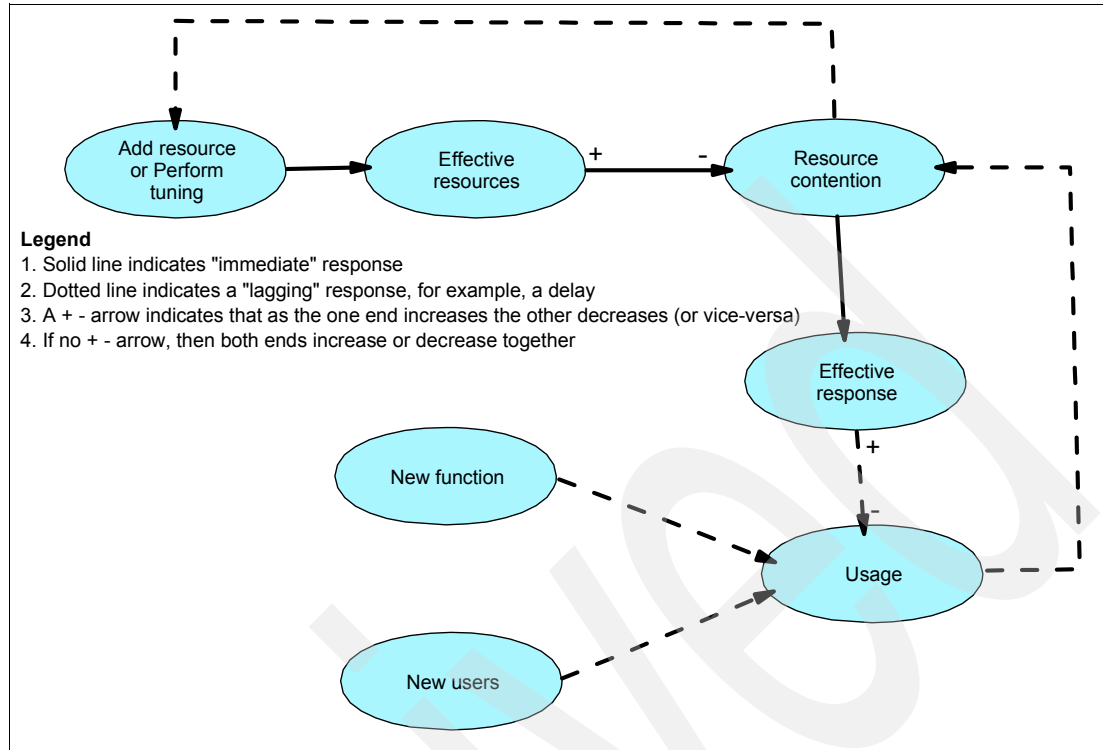


Figure 10-4 Capacity planning: Cause-effect graph

When tracing the path of one loop, the following actions occur:

- ▶ When new users are added to the system, after a delay, usage tends to rise.
 - As a result of the increased usage, there is 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, although small 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.

10.5 Generic performance process and methodology

This section reviews the overall process of performance monitoring and capacity planning, which is shown in Figure 10-5. The perspective is generally from that of a performance specialist.

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 is most likely generated from the performance architect or specialist. This does not absolve developers from being aware of the performance aspects of their design and coding.

See Chapter 9, “Java and WebSphere application design” on page 285, for more information about specific design and coding techniques that you should be aware of that can improve your application’s performance.

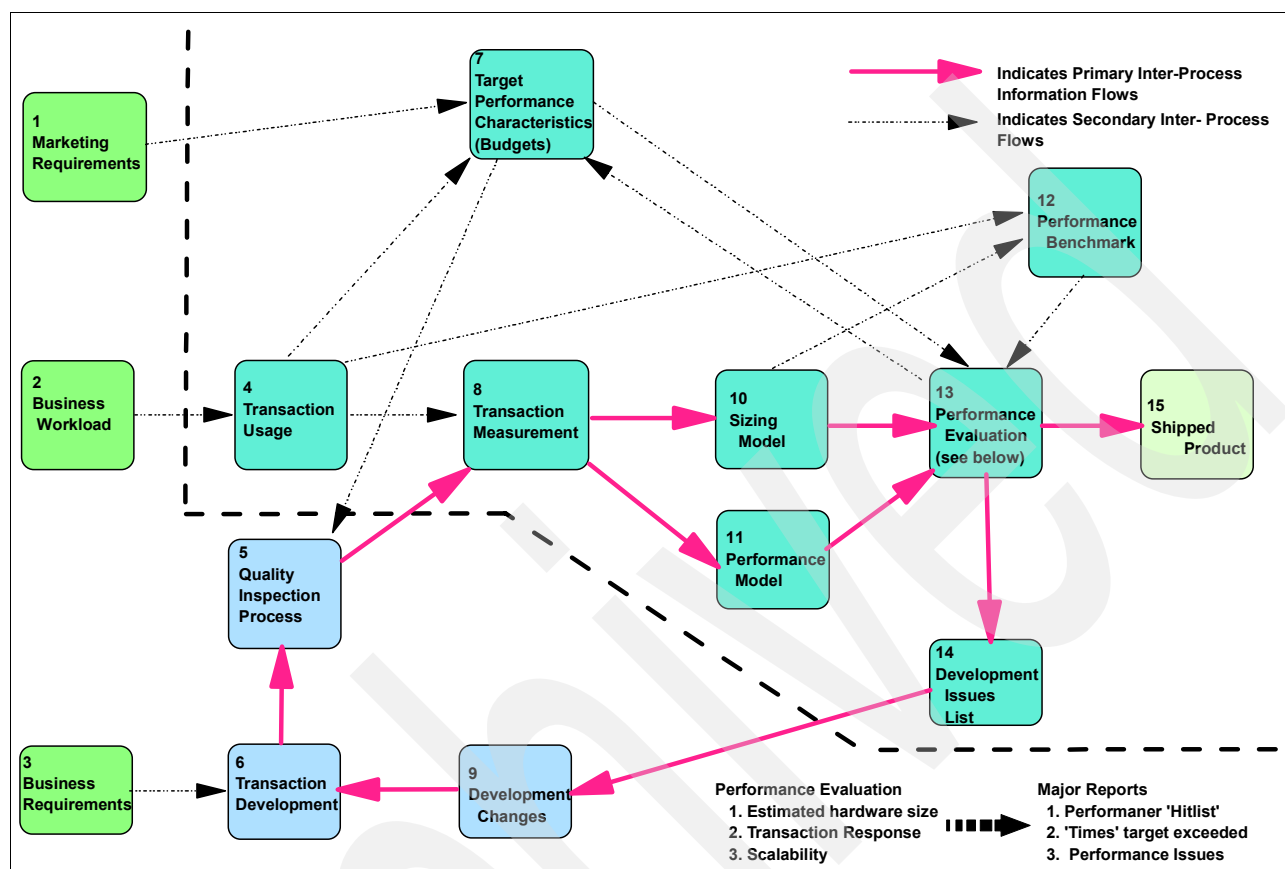


Figure 10-5 Capacity planning: generic representation of the performance process

If your application has been well-designed and implemented in a performance-conscious manner, there should be little need for significant tuning effort. Now we outline the performance methodology that was used successfully in a development organization. Overall you can categorize the various process elements as:

- ▶ **Macro:** At the highest level of granularity, either a business transaction or a machine transaction
- ▶ **Micro:** At the machine transaction executable statement level

There is an essential feedback process with the following steps:

1. Set resource (performance) requirements.
2. Measure results.
3. Evaluate targets and assess the impact of changes
4. Plan changes.
5. Implement changes and, if required, re-set requirements.

10.5.1 Process steps

Table 10-2 describes each of the steps in the previous list in greater detail. The more important process steps are illustrated in Figure 10-5.

Table 10-2 Capacity planning: detailed description of process steps

| Step | Title | Description |
|-----------|--|---|
| 4 | Analyze Transaction Usage (or Flows) | Starting from a defined business workload, termed the “reference workload”, map to the individual machine transactions that implement the business transactions. 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% or more of the workload. |
| 7 | Target Performance Characteristics | Set development budgets based on primary input documents that may include: <ul style="list-style-type: none"> ► Non-functional requirements ► Defined physical processes ► Business Transaction to Machine Transaction flows ► Tests and measurements carried out as part of preliminary analysis |
| 8 | Measure Server Transaction Characteristics | Using server monitoring tools, measure machine transaction resource use. Based on these measurements, update your sizing model to provide a “how-goes-it” evaluation. |
| 10 and 11 | Sizing and Performance Models | Create 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. |
| 12 | Performance Benchmark | Measure how a simulated live load will run in a defined environment. Interim mini-benchmarks and stress testing runs provide early indications prior to a fully functional benchmark. |
| 13 | Performance Evaluation (Evaluate Server Transaction Characteristics) | Based on sizing targets, evaluate the transactions and determine the need for specific transactions to improve performance. Consider both design and implementation. Detailed measurements (at the executable statement level) may be required in some 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. Extract management feedback. |
| 14 and 9 | Define Improvement Strategy/Strategies | To the development team or teams, give the measurement results and suggested transaction changes that need to be implemented to close in on the target figures. |

It is a fundamental precept of performance evaluation and tuning that *the effort is placed where maximum gain will be achieved*. Therefore, consider the specific tuning in the context of the whole application, not only at an individual component level.

Given a set of measurements and the granularity provided by the tool used, compare the derived values to your established targets. Then, you must assess the potential improvements resulting from making changes. There must be a balance between the effort required to measure and implement the changes and the 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 worthwhile 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 improvements can be gained by making a change? And what is the impact of that change on the work completed?
- Are there any common factors that apply to all transactions, indicating that common code could or should they be changed?

Changes may apply to:

- **Design:** In which case a rewrite may be required
- **Reuse:** Whether appropriate, in which case the implementation needs to be modified
- **Code:** Again implementation needs to be modified

Figure 10-6 shows the performance activity profile. It illustrates the performance resources that may likely be required during the life of a development project.

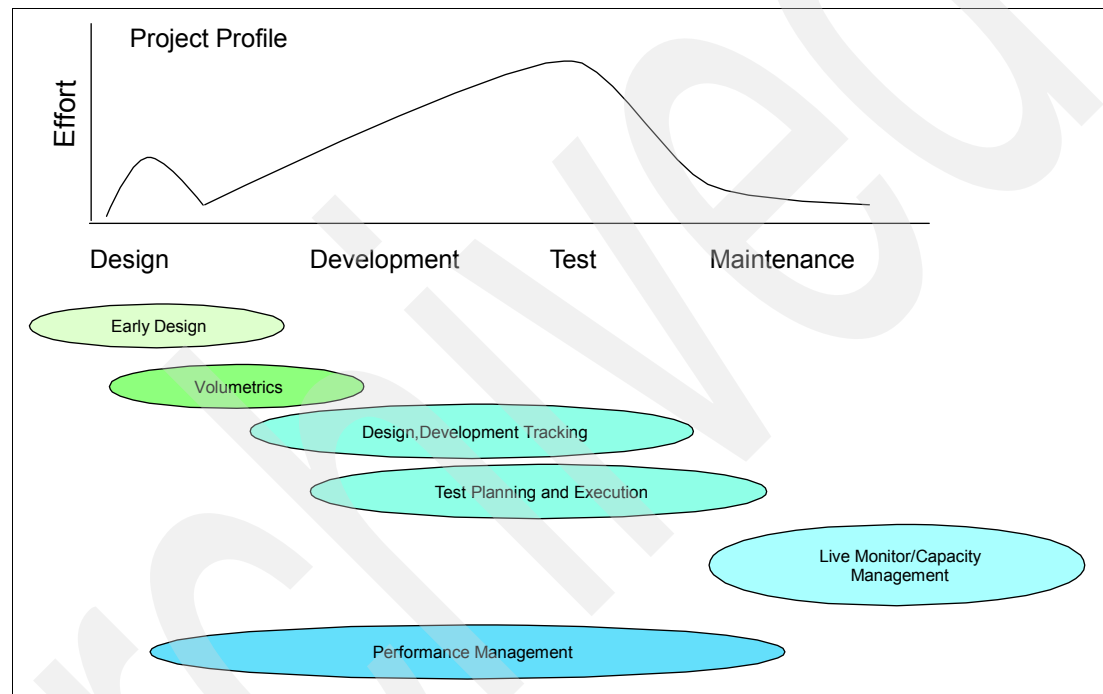


Figure 10-6 Capacity planning: Performance activity profile

10.6 Capacity planning and workload estimation tools

There are no easy answers when it comes to capacity planning questions. Use the IBM Workload Estimator for iSeries to size an iSeries server for the WebSphere Application Server. You should also refer to the corresponding publication *iSeries Performance Capabilities Reference Version 5, Release 2, SC41-0607*.

The iSeries deployment environment is rich in choices. iSeries developers who want to deploy applications using the WebSphere Application Server can make choices between using WebSphere functions (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 instead of standard planning tools.

You can use the Performance Management tools (see 4.3.5, “Performance Management for iSeries, PM/400, and PM eServer iSeries” on page 100) and the Workload Estimator capacity planning tools to plan for future resource demands on system performance. In particular, PM eServer iSeries allows you to easily submit your utilization and growth data to the Workload Estimator. Use the Workload Estimator to size your next upgrade.

10.6.1 IBM Workload Estimator

The IBM Workload Estimator tool provides IBM, IBM Business Partners, and IBM Clients with a comprehensive iSeries sizing tool that investigates deployment of new workloads in a stand-alone manner or in combination with current workloads. The recommendations provided by this tool are for the most recent hardware. They project the model, CPU% utilization, memory, disk arms, and capacity.

Using IBM Workload Estimator is easy. There are less than a dozen questions per application. 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 on the proposed hardware. A printed report is available that provides defaults, input parameters, and results.

To launch the Workload Estimator, point your browser to:

<http://www-912.ibm.com/supporthome.nsf/document/16533356>

You can run the Workload Estimator online at the IBM Web site or locally on a machine within your organization, as shown in Figure 10-7.

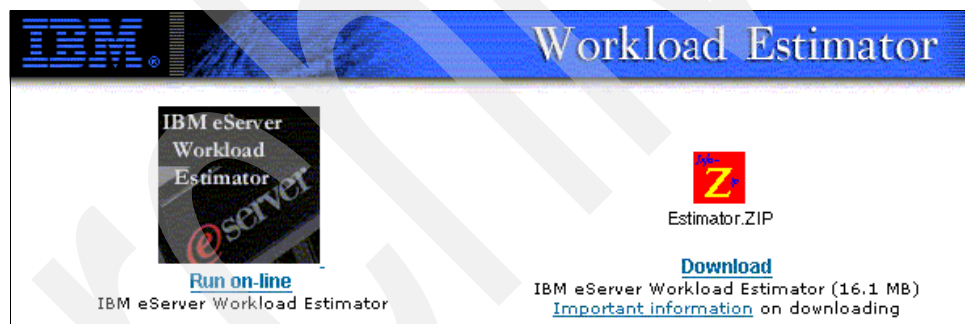


Figure 10-7 Capacity planning: Accessing the Workload Estimator

If you decide to run the Workload Estimator on the IBM Web site, click the **Run on-line** link, which routes you to the following URL:

<http://www-912.ibm.com/estimator/index.html>

If you download the Workload Estimator, you need to follow the installation instructions, which include installing a supported JDK and Jakarta Tomcat prior to installing Workload Estimator. Be sure to follow the installation instructions carefully. Otherwise, the Workload Estimator will not work. Assuming you have succeeded in your installation, the Workload Estimator is then available on:

`http://machine:port/wle/EstimatorServlet`

Here *machine* indicates the server which you have deployed Tomcat, and *port* equals the port number that you have assigned. The default port is 8080.

Using the Workload Estimator is straightforward. It requires a minimum amount of information and may be completed in a few steps as summarized in the following list.

1. Select the workload type.

Figure 10-8 shows the workload selection page for Workload Estimator. You may specify multiple workloads. But remember that, during this process, you must describe each and every workload separately.

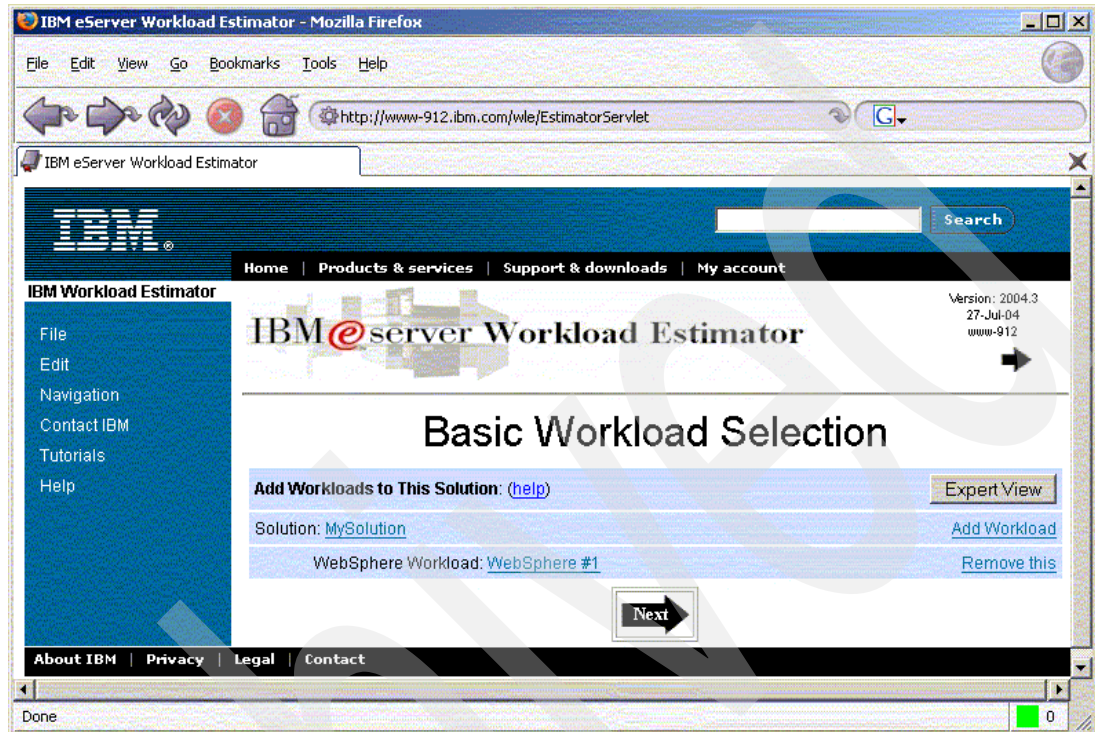


Figure 10-8 Capacity planning with IBM Workload Estimator: Workload selection

2. Enter the workload details

Figure 10-9 shows you can enter the specified workload volume and detail distribution. This example shows only the WebSphere workload details, but you must describe all the workloads that you specified on the previous page.

IBM eServer Workload Estimator - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://www-912.ibm.com/wle/EstimatorServlet

IBM eServer Workload Estimator

IBM

Home | Products & services | Support & downloads | My account

IBM Workload Estimator

File
Edit
Navigation
Contact IBM
Tutorials
Help

IBM eServer Workload Estimator

Version: 2004.3
27-Jul-04
www-912

WebSphere #1

WebSphere Workload Definition

A visit is a group of transactions from a given user.

1. How many [visits per hour](#) are anticipated during the **busiest** hour of the day?

In a typical visit, how many of the following operations will occur:

2. [Static web pages and files](#) served?

3. Java Server Pages ([JSPs](#)) served?

4. [Java Servlets](#) executed?

Select [Advanced Configuration](#) button below to perform optional advanced configurations options.

[Back](#) [Advanced Configuration](#) [Next](#)

Done

Figure 10-9 Capacity planning with IBM Workload Estimator: Workload definition

3. Specify the workload.

Figure 10-10 shows how you can specify the complexity of the WebSphere workload.

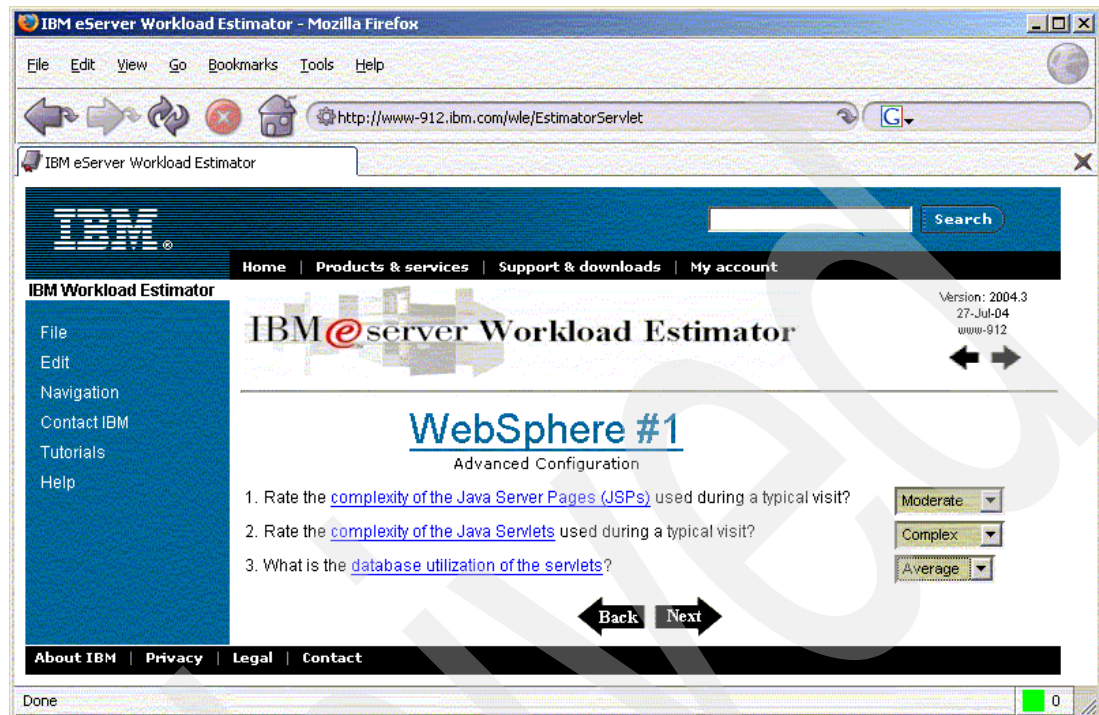


Figure 10-10 Capacity planning with IBM Workload Estimator: Specifying workload complexity

4. Review the proposed configuration

Figure 10-11 shows the proposed configuration that you need to support the previously described workload. Remember that the more accurately you describe the application functions and the workload, the more accurately the iSeries configuration is sized.

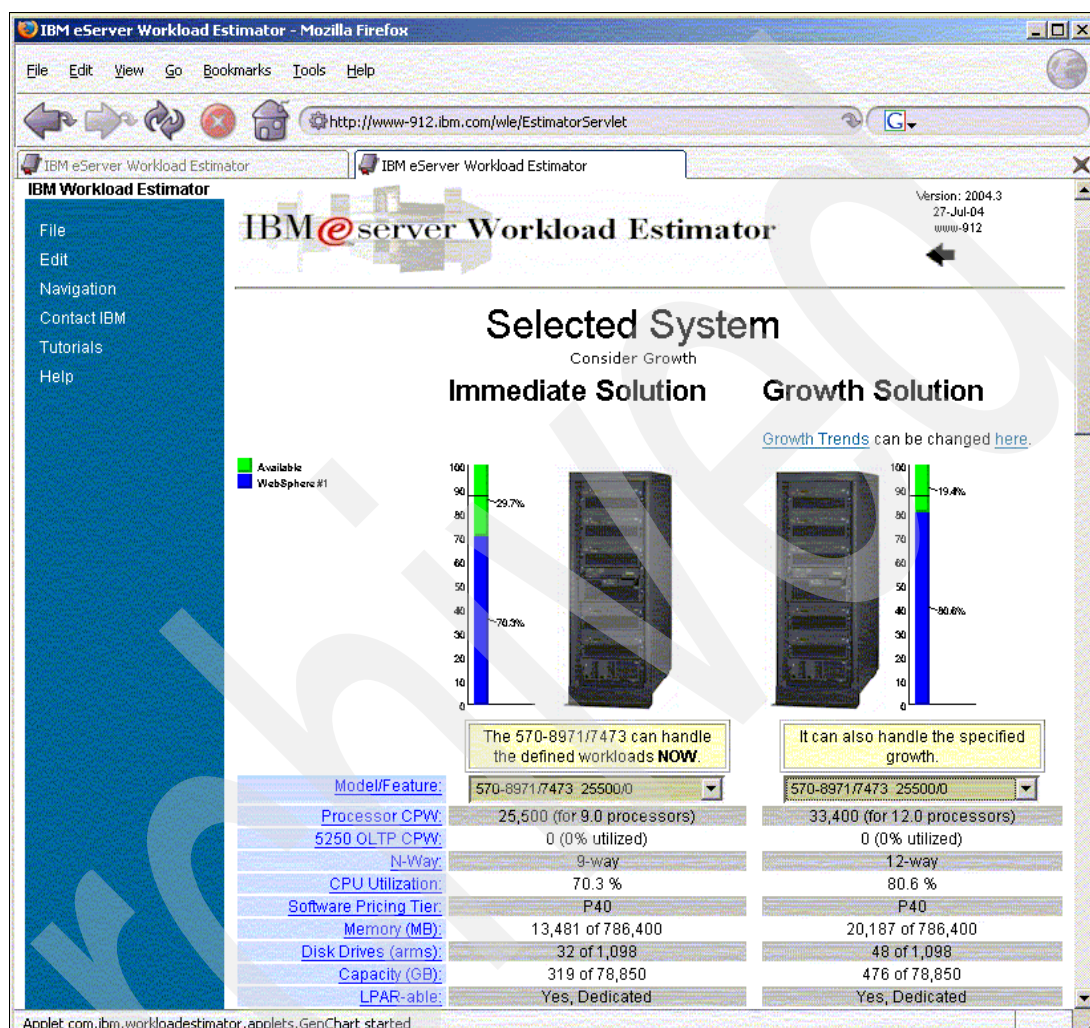


Figure 10-11 Capacity planning: IBM Workload Estimator: Estimated configuration

Sensitivity to variations

By making simple changes to the details already specified, such as 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 your system is sized.

10.6.2 PM eServer iSeries

PM eServer iSeries is a set of automated tools that collect performance data on your iSeries server and provide reports that are easy to understand. It helps you determine the long-term workload growth and predict when a system upgrade may be needed. You also have the possibility to use the data collected with PM eServer iSeries as input to IBM Workload Estimator. For a full description of PM eServer iSeries services and features, see:

<http://www-1.ibm.com/servers/eserver/iseriess/pm>

You only have to know how to start PM eServer iSeries. Then, when it is started, it collects the proprietary system performance data on your iSeries server. Learn more about PM eServer iSeries in 4.3.5, “Performance Management for iSeries, PM/400, and PM eServer iSeries” on page 100.

Tip: The data collected with PM eServer iSeries provides excellent input for Workload Estimator.

10.7 Disk arm sizing

Insufficient number of disk arms can decrease the system performance. Use the disk arm sizing guidelines described in *iSeries Disk Arm Requirements*, on the Web at:

<http://www-1.ibm.com/servers/eserver/iseries/perfmgmt/diskarm.htm>

10.8 Stress testing WebSphere applications

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 stress testing. Several tools, either open-source or commercial, can help you with this type of testing. Typically, such tools allow you to capture client interactions with an application. Then, you can save this information, and replay it in a manner which simulates multiple *virtual* users. Learn more about stress testing in 4.7, “Stress testing tools” on page 157.

Note: The effort required to create a representative workload is significant. Do not approach this task approached lightly.

10.9 Sizing an application under development

While it is relatively straightforward to determine apparently realistic numbers to enter into a tool, such as IBM Workload Estimator, there is always a lingering doubt as to the way the application will perform in service in terms of response, throughput, and capacity. We recommend that you use an ongoing measurement process during the full development life cycle. We extend this to outline a method where the first capacity estimates may be validated as development proceeds.

Having tools such as the Tivoli Performance Viewer (see 4.5.3, “Tivoli Performance Viewer” on page 130) and PTDV, both of which provide measurement of CPU time used, you can use this data over the course of the development life cycle. You can take multiple measurements to obtain a valid estimate of production sizing requirements.

The method that we suggest is summarized here:

1. Document the targeted business workload. It will not be correct, but you can refine it 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 resources 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 I/O is not particularly difficult.
4. From the mappings, 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. You are now in a position to provide specific feedback to the transaction developers who have written code with characteristics exceeding the norm.

While simple to define, this method requires some effort to implement. After the initial setup is complete, regular updating of transaction characteristics will allow easy assessment of the current development status against the sizing goals.

Archived

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/SG246383>

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG246383.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

| <i>File name</i> | <i>Description</i> |
|----------------------|--|
| t3install.zip | This file contains the Trade3 application installation code. |
| trade51db.zip | This files contains the Trade51DB database with relevant data. |

System requirements for downloading the Web material

The following system configuration is recommended:

| | |
|--------------------------|------------------------|
| Hard disk space: | 20 MB minimum |
| Operating system: | Windows 2000 or XP |
| Processor: | Pentium® III or better |
| Memory: | 512 MB |

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip files into this folder.

Open the InstallationInstructions.pdf file and follow the instructions to install the Trade3 application and associated database to your iSeries.

You should have WebSphere Application Server V5.1 Base Edition on your iSeries system in order to run Trade3.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 374. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *Advanced Functions and Administration on DB2 Universal Database for iSeries*, SG24-4249
- ▶ *IBM WebSphere V5.1 Performance, Scalability, and High Availability WebSphere Handbook Series*, SG24-6198
- ▶ *WebSphere Application Server V5 for iSeries: Installation, Configuration, and Administration*, SG24-6588
- ▶ *AS/400 HTTP Server Performance and Capacity Planning*, SG24-5645
- ▶ *HTTP Server (powered by Apache): An Integrated Solution for IBM @server iSeries Servers*, SG24-6716
- ▶ *AS/400e Diagnostic Tools for System Administrators: An A to Z Reference for Problem Determination*, SG24-8253

Other publications

These publications are also relevant as further information sources:

- ▶ *Performance Tools for iSeries*, SC41-5340
- ▶ *iSeries Performance Capabilities Reference Version 5, Release 2*, SC41-0607
<http://www-1.ibm.com/servers/eserver/series/perfmgmt/resource.htm>

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ Planning for availability in the enterprise
http://www-106.ibm.com/developerworks/websphere/techjournal/0312_polozoff/polozoff.html
- ▶ WebSphere Application Server for iSeries Information Center Version 5.1
<http://publib.boulder.ibm.com/was400/51/english/index.htm?info/rzaiz/51/was.htm>
- ▶ Performance Management for IBM @server iSeries
<http://www-1.ibm.com/servers/eserver/series/perfmgmt/resource.htm>
- ▶ iDoctor for iSeries
http://www.ibm.com/eserver/series/support/i_dir/idoctor.nsf

- ▶ iSeries Information Center V5R2
<http://publib.boulder.ibm.com/series/v5r2/ic2924/index.htm>
- ▶ IBM Tivoli Web Site Analyzer
<http://www-306.ibm.com/software/tivoli/products/web-site-analyzer/>
- ▶ AWStats Web Site
<http://awstats.sourceforge.net>
- ▶ Software Knowledge Base
http://www-912.ibm.com/s_dir/slkbase.nsf/slkbase
- ▶ IBM Rational Software
<http://www.ibm.com/software/rational>
- ▶ OpenSTA Web Site
<http://www.opensta.org>
- ▶ IBM @server iSeries Support
<http://www-912.ibm.com/>
- ▶ Developing and Deploying Command Caching with WebSphere Studio V5
http://www-106.ibm.com/developerworks/websphere/registered/tutorials/0306_mcguinnes/mcguinnes.html
- ▶ Apache Web Site
<http://www.apache.org>
- ▶ IBM @server Workload Estimator
<http://www-912.ibm.com/estimator/index.html>

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

Numerics

5722-JV1 218

A

- access bean 328
- access intent 326
- access log 147, 209
- access plan 299
- ACID 324
- active request 232
- Active Size column 339
- active thread 173
- activity level 177, 181
- ADDENVVAR command 221
- additional number of active jobs 172
- additional number of total jobs 171
- ADDPEXDFN command 227
- administration 18
- administrative console 311
- Advisor 99
 - conclusions 99
 - recommendations 99
- Agent Controller 118
- allocation system values 169
- Allow Overflow 245
- Analyze Java Virtual Machine (ANZJVM) command 101
- AnyNet 190
- ANZJVM command 25, 101
- AOA (application-oriented architecture) 1
- Apache JMeter 158
- application 2
 - profiling 330
 - stress testing in WebSphere 368
- application development 18
- Application Profiler 230
- application scope 316
- application-oriented architecture (AOA) 1
- architecture xi, 386
 - emerging 8
 - service-oriented 8
- ArrayList 293
- ASP
 - capacity 29
 - usage 29
- assistance level 170
- asynchronous I/O 204
- asynchronous operating mode for GC 219
- atomic 324
- AutoCommit 301
- automatic tuning 168
- availability 355
- average base time 334
- AWStats 148

B

- balance tree 293
- base storage pool activity level 173
- base storage pool minimum size 174
- base time 334
- batch immediate 219
- batch insert 303
- BCI 219
- Bean Scripting Framework (BSF) 8
- bean-managed persistence (BMP) 327
- Beans.instantiate() 312
- BigDecimal 303
- block size 306
- blocking READ_ONLY, WRITE_ONLY 308
- BMP (bean-managed persistence) 327
- bottleneck 326
- browser 5
- brute force method 7
- BSF (Bean Scripting Framework) 8
- bucket 264
- buckle zone 235
- business-to-business sites 348
- bytecode 218

C

- cache monitor 253
- cache replication 257
- cache size 264
- cachespec.xml 250
- caching 200, 352
 - a data source 307
- Caching class 319
- call level interface (CLI) 296
- call stack 53
- calls 334
- capacity 22
- capacity planning 6, 345, 358, 362
- cause-effect 358
- CCSID 303
- CCSID 13488 303
- Change Prestart Job Entry (CHGPJE) command 187–188
- Change Routing Entry (CHGRTGE) command 177
- Change Subsystem Description (CHGSBSD) command 176
- Change System Value (CHGSYSVAL) command 169
- CHGPJE command 187–188
- CHGRTGE command 177
- CHGSBSD command 176
- CHGSYSVAL (Change System Value) command 169
- child job, multi-threaded 204
- class interactions 127
- class statistics 127
- class variables 289

- cleanup interval 265
- CLI (call level interface) 296
- client
 - considerations 4
 - Internet Explorer 4
 - Mozilla Firefox 4
 - Netscape Navigator 4
 - performance 4
- closed queue 232
- clusters 10
- CMP (container-managed persistence) 325
- collecting performance data 93
- collection 292, 301
 - lists 293
 - maps 293
 - sets 293
- Collection Services 25, 36
- Collections Framework 292
- com.ibm.as400.access.AS400JDBCdriver 295
- com.ibm.CORBA.MaxOpenConnections 268
- com.ibm.db2.jdbc.app.DB2Driver 295
- command framework 316
- communication lines 7, 100
- communications IOPs 7
- communications trace 25, 156
- computational tasks 10
- concatenation 287
- concurrent garbage collection 15
- connection 299
- connection pooling 258, 352
- consistent 324
- container-managed persistence (CMP) 325
- cost 355
- CPU 7
- CPU utilization 99
- CPW 23
- Create Performance Data (CRTPFDRDTA) command 96, 98
- CRTJVAPGM command 221
- CRTPFDRDTA command 96, 98
- CUM PTF package 166
- cumulative time 334
- Current Activity folder 52
- customer self-service sites 348

D

- Data Class Access Bean 328
- data conversion 303
- data primitives 289
- data source 258, 298, 312
 - caching 307
- data transfer object 328
- data wrapper 328
- database 294
- Database Monitor 85
- Database Monitor for iSeries 85
- database server 2
- database tuning 297
- data-related activity 10
- DB Monitor 85

- DB2 UDB for iSeries 353
- DB2 UDB stored procedure 8
- DBMonitor (DBMON) 25
- DDM (Distributed Data Management) 294, 308
- DE (direct execution) 13
- deadlock, avoiding with datasource connections 261
- Default settings for JVM 223
- Delete Communications Trace (DLTCMNTRC) command 156
- denial of service (DoS) 211
- direct execution (DE) 13
- direct processing 220
 - performance impact 223
- directive, ThreadsPerChild 205
- dirty flag 328
- dirty read 304
- disk 100
 - protection status 30
- disk arm 7
 - sizing 368
- disk IOPs 7
- disk offload 255
- disk space 7
- Display Active Prestart Jobs (DSPACTPJ) command 184
- Display Job Tables (DSPJOBTL) command 170
- Display Performance Data (DSPPFDRDTA) command 98
- Distributed Data Management (DDM) 294
- Distributed Program Call (DPC) 294, 309
- distributed session 247
- DMPJVM (Dump Java Virtual Machine) command 101
- DMPJVM command 25
- DoS (denial of service) 211
- DPC (Distributed Program Call) 294, 309
- DSPACTPJ (Display Active Prestart Jobs) command 184
- DSPDTAARA display 167
- DSPJOBTL command 170
- DSPPFDRDTA (Display Performance Data) command 98
- dspwasinst script 357
- Dump Java Virtual Machine (DMPJVM) command 101
- Dump User Trace (DMPUSRTRC) command 151
- durable 324
- Dynamic Cache Policy Editor 252
- dynamic cache service 247
- dynamic caching 200–201
- dynamic changes 347
- dynamic SQL 296–297

E

- EBCDIC 303
- edge cache statistic 255
- Edge Side Include (ESI) 255
- EJB
 - coding considerations 316
 - container 264
 - homes 319
 - isolation level 325
 - stubs 357
 - stubs, WLM 357
 - workload management 357
- EJB 1.1 267

- EJB 2.0 267, 325
- ejbCreate() 323
- ejbPasivate() 323
- ejbRemove() 323
- End Communications Trace (ENDCMNTRC) command 156
- End Database Monitor (ENDDDBMON) parameters 87
- ENDTCPSVR command 150
- enterprise beans 8
- entity bean 316, 328
- error log 146, 209
- ESI (Edge Side Include) 255
- evaluating stress testing tools 158
- exception 290
- execution flow 127
- execution mode 219
- execution time 117
- Expert Cache 175, 183
- extended dynamic SQL 297
- extended dynamic support 296
- Extensible Markup Language (XML) 8
- external caching 257

F

- failover 1, 346
- Fast Response Cache Accelerator (FRCA) 203, 257
- fault isolation 347
- fault-tolerance 346
- flow control 210
- FRCA (Fast Response Cache Accelerator) 203, 257
- Functional Tester for Java and Web 159

G

- garbage collecting threshold 15
- garbage collection 15, 102, 127, 223, 288
 - basics 15
 - concurrent 15
 - cycle 224
 - instantiation results 288
 - performance tuning 224
 - pitfalls 16
 - problems 230
 - stop and copy 15
 - stop and copy method 223
- garbage collector 15, 219, 289, 337–338
- GC (garbage collector) 15
- get...(columnIndex) 301
- get...(columnName) 301
- Global Transaction 281
- global variables 292
- Grinder 158
- group PTF 166

H

- hardware 5, 7
- HashMap 293
- HashSet 293
- heap 102

- Heap Analysis Tools for Java 55
- Heap Analyzer 55
- HEAPANA command 25, 81, 101
 - parameters 81
- heavy load zone 235
- high availability 1
 - fail over 1
 - load balancing 1
- Home Locator class 319
- home method 327
- horizontal scaling 355
- host interactions 128
- HostNameLookups 210
- HTTP
 - access logs 147
 - administration interface 149
 - Collection Services 152
 - error logs 146
 - server tracing 149
 - status codes 148
- HTTP performance tools 145
- HTTP server 2
 - tuning options 204
- HTTP Server (powered by Apache) 199
- HTTP tracing 25
- HttpServlet Init() method 310
- HttpSessions 309–310
- Hypertext Transfer Protocol (HTTP) 8

I

- IBM Developer Kit for Java driver 295
- IBM HTTP Server (powered by Apache) 199
- IBM network dispatcher (eND) technology 352
- IBM Rational Robot 159
- IBM Rational Test Manager 159
- IBM Toolbox for Java 17, 294
 - driver 295
- IBM Toolbox JDBC driver 183
- IBM Workload Estimator 363
- iDoctor 25, 81
- inactive pool cleanup interval 265
- include directive 315
- ineligible thread 173
- info beans 321
- init() 310
- initial heap size parameter 15, 224
- initial number of active jobs 172
- initial total number of jobs 169
- instance statistics 126
- instantiation 288
- interpretation 218
- interpreted mode 220
- Invalidation Time-out 247
- IOPs 100
- Is Growable property 270
- iSeries xi, 23, 354, 386
 - performance behavior 166
 - performance measurement tools 25
 - scaling considerations 353
- iSeries Developer Kit for Java 296

- isolated 324
- isolation level 304
 - for EJBs 325

J

- Java xi, 386
 - compilation environment 12
 - compiling and running 14
 - execution environment 12
 - general design 286
- Java beans 8
- JAVA CL command 218
- Java Database Connectivity (JDBC) 294
- Java Developer Kits 218
- Java execution mode 219
 - direct processing 220
 - interpreted 220
 - JIT 220
 - just-in-time compiler 220
 - just-in-time compiler and direct processing 220
- Java garbage collection 11, 15
- Java Messaging Service (JMS) 8
- Java Naming and Directory Interface (JNDI) 310
- Java Native Interface (JNI) 293
- Java performance 7
- Java Runtime Environment (JRE) 12, 219
- Java runtime modes of execution 219
- Java thread priority 189
- Java transformer 13, 220
- Java virtual machine (JVM) 2, 12, 353
- java.compiler 221
- java.rmi.RemoteException 325
- java.util.logging 292
- java.version 218
- JAVAGC indicator 228
- JAVAGCTOOLS 101, 106
 - options 111
- javax.servlet.http.HttpSession.invalidate() 247
- JDBC (Java Database Connectivity) 294, 296, 298, 314, 354
- JDBC driver limitations 358
- JDBC isolation level 304
- JDK 1.4 218
- JDKs 218
- JIT
 - compiler 13
 - performance impact 223
- JMS (Java Messaging Service) 8
- JMS resources 281
- JNDI (Java Naming and Directory Interface) 310
- JNDI names 317
- JNDI naming lookup 313
- JNI (Java Native Interface) 293
- job log 210
- Job Watch
 - job specific 42
 - system wide 42
- Job Watch Trigger Support 85
- Job Watcher
 - trigger support 85

- viewing graphs 53
- viewing reports 53
- job-specific collection 42
- JRE (Java Runtime Environment) 12, 219
- JSP batch compiler 275
- JSP coding consideration 309
- just-in-time compiler 220
 - and direct processing 220
- JVM (Java virtual machine) 2, 12, 353
- JVM heap 219

K

- KeepAliveTimeout 210

L

- lazy initialization 288
- light load zone 235
- LimitRequestBody 211
- LimitRequestFields 211
- LimitRequestFileSize 211
- LimitRequestLine 212
- LimitXMLRequestBody 211
- LinkedList 293
- list interface 293
- listener port 280
- listener service 279
- lists 293
- load balancing 346
- load distribution 159
- load testing 157
- local caching 201
- local interface 267, 323
- local variables 292
- Log Analyzer 25, 117, 142, 148
- logging 25, 208, 291
- lookup 313
- loops 289

M

- machine interface (MI) 218
- machine storage pool size 174
- main memory 7, 174
- main storage 7, 100, 174
- maintainability 355
- Management Central 25, 36
- management collection 98
- manual tuning 168
- map interface 293
- maps 293
- Max Application Concurrency value 236
- maximum activity level of system 174
- maximum connection pool 259
- maximum heap size 15
- Maximum in-memory session count 245
- maximum number of jobs 171
- maximum throughput value 235
- MaxKeepAliveConnections 232, 239, 241
- MaxKeepAliveRequests 232, 239, 243

- MDB (message-driven bean) 327
- measurement column 336
- memory 5, 309, 332, 339
- memory leak 102, 105, 223, 337, 341
- memory pool 174
 - creating 174
 - determine size 177
 - private 174
 - shared 174
 - sizes 177
- memory utilization 117, 126
- message concurrency 281
- message listener service 279
- message-driven bean (MDB) 327
- method inlining 291
- method invocation 125
- method resolution 290
- Method Statistics view 333
- methodology 21
 - performance management 21
- minimum connection pool 260
- mixed configuration 347
- Mixed Mode Interpretation (MMI) 222
- MMI (Mixed Mode Interpretation) 222
- mod_deflate 208
- multi-row session 247
- multi-threaded child job 204

N

- native driver 295–296
- native JDBC driver 183, 357–358
- network components 5
- network design 5
- network dispatcher (eND) technology 352
- Network File Cache (NFC) 203
- network performance 5
- NFC (Network File Cache) 203
- nonrepeatable read 304

O

- object 15
- object instantiation 288
- object interactions 127
- object reference 127
- object table dump 56
- Objects Instantiation 288
- ODBC environment 295
- OLTP (online transaction processing environment) 297
- online shopping sites 348
- online transaction processing environment (OLTP) 297
- open and close 308
- open queue 232
- open source stress testing tool 158, 162
- open-source 368
- OpenSTA 158, 162
- operating system 5, 23, 166
- Operations Navigator 95
- optimistic read 325
- optimistic update 325

- OPTIMIZE parameter 221
- OS/400 system tuning 168
- os400.defineClass.optLevel 221
- os400.jit.mmi.threshold 223
- outer transaction context 316

P

- package statistics 127
- Package Statistics view 338
- page fault 175, 177
- page scope 316
- paging 177
 - CALC 183
 - FIXED 183
- parallelism 351
- parsing number 289
- pass-by-reference 267
- pass-by-value 267
- Percent Maxed 240, 266
- performance xi, 355, 386
 - activity profile 362
 - application design 1
 - best practices 1
 - cause-effect 358
 - coding standards 1
 - database 25
 - emerging architectures 8
 - HTTP 25
 - iSeries 166
 - Java 25
 - measurement 6
 - network design 5
 - problem determination process 24
 - resource consumption 22
 - strategy 22
 - system 25
 - tools 21, 25
 - trend analysis 22
 - tuning 21
 - virtualized deployment 1
 - Web server 200
 - WebSphere 25
- performance activity profile 362
- Performance Advisor in Tivoli Performance Viewer 133, 139
 - Using 139
- Performance Advisors 25, 117, 132
- performance data 96
- performance goals 22
- performance measurement tools 25
- performance methodology 359
- Performance Monitor Services 129
- Performance Monitoring Infrastructure (PMI) 25, 128
- performance objectives 22
- performance process 359
- performance strategy 23
- Performance Tools for iSeries 25, 98, 154
- Performance Trace Data Visualizer (PTDV) 230, 368
- persistent connection 204, 242
- persistent session 357

- pessimistic read 325
- pessimistic update 325
- pessimistic update - Weakest Lock At Load 325
- PEX Analyzer 73
- PFRDTA 16
- phantom read 304
- PM eServer iSeries 363, 367
- PM/400 367
- PMI (Performance Monitoring Infrastructure) 25, 128
- pool 174
- port 357
- ports 98
- Pre-compile JSP 274
- preCompileJSPs 274
- prepared statement 299
- prestart job 183
- Print Communications Trace (PRTCMNTRC) command 156
- Print PEX Report (PRTPEXRPT) command 227
- private memory pool 174
- problem determination 24
- process interactions 128
- processor speed 5
- Profile on server window 331
- profiling 23, 25, 117
 - application 330
 - applications 118
 - data 16
 - filters 332
 - on the iSeries 16
- protection status of disk 30
- proxy 5
- proxy caching 201
- proxy service 201
- PRTPEXRPT (Print PEX Report) command 227
- PTDV (Performance Trace Data Visualizer) 230, 368
- PTF package 166
- publish/subscribe Web sites 348

Q

- QACTJOB 169, 172
- QADLACTJ 169, 172
- QADLSPLA 169
- QADLTOTJ 169, 171
- QAQQDBMN 86
- QBASACTLVL 172–173
- QBASPOOL 172, 174
- QCMN subsystem 183
- QJOBMSGQFL 169
- QJOBMSGQMX 169
- QJOBMSGQSZ 169
- QJOBMSGQTL 169
- QJOBSPLA 169
- QMAXACTLVL 172, 174
- QMAXJOB 169, 171
- QMAXSPLF 169
- QMCHPOOL 172, 174
- QPFRADJ 168
- QRCLSPLSTG 169
- QSERVER subsystem 183

- QSQSRVR 183, 188
- QSQSRVR jobs 187
- QTOTJOB 169
- QZDASOINIT 183, 295
- QZRCRSVS 295

R

- RAM (Random Access Memory) 174
- Random Access Memory (RAM) 174
- read committed 304
- read uncommitted 304
- READ_ONLY 308
- read-ahead schemas 327
- ReadAll() method 308
- record-level access 295, 308
- Redbooks Web site 374
 - Contact us xiii
- remote interface 267, 323
- remote method calls 316
- repeatable read 304
- request scope 316
- resizable array 293
- resource scavenging 10
- response time 342
- ResultSet 314
- reverse proxy 201
- Robot tool 158
- rollback 302
- Run/wait signature graph 52
- RUNJAVA command 218, 221
- RUNPTY setting 188
- Runtime Performance Advisor 132–133
 - using 133

S

- saturation point 234
- scalability 346, 349, 355
 - objectives 346
- scaling 345
- scaling technique 350
- scaling, iSeries 353
- script
 - dspwasinst 357
 - recording 159
- scrollable result sets 305
- Secure Sockets Layer (SSL) 204
- security 19, 347
- semi-automatic system tuning 168
- serializable 304, 326
- serialize session access 246
- server 7, 200
- server components 6
- server performance 6
- server tracing, HTTP 149
- service broker 8
- service provider 8
- service requester 8
- service-oriented architecture (SOA) 1, 8
- servlet clustering 357

- servlets 309
- session bean 316
 - stateful 324
- session scope 316
- set interface 293
- setEntityContext() method 328
- sets 293
- shared memory pool 174
- short circuit 290
- SingleThreadModel 310
- sizing an application 368
- snapshot 102
- SOA (service-oriented architecture) 1, 8
- SQL extended dynamic support 296
- SQL package 296
- SQL performance monitor data 87
- SQL statement cache 296
- SQL Visual Explain 25, 91
- SSL (Secure Sockets Layer) 204
- SST (System Service Tools) 106
- Start Communications Trace (STRCMNTRC) command 156
- Start Performance Tools (STRPFRT) command 94
- Start TCP/IP Server (STRTCPSVR) command 149
- stateful session bean 328
- Statement 299
- statement cache 297
- statement cache size 261
- static caching 200–201
- static content 315
- static content location 212
- static SQL 297
- stop and copy garbage collection method 15, 223
- stop-and-copy operating mode for GC 219
- storage pools 174
- storage system values 172
- stress testing 23, 164, 368
 - IBM Rational 158
 - open source tool 158
 - tool resources 164
 - tools 157–158
 - tools for load distribution 159
- String 287
 - creation 286
 - manipulation 286
- StringBuffer 287
- STRPFRT (Start Performance Tools) command 94
- STRSST 106
- STRSST GC tools 25
- Summarized Graphs folder 52
- Summarized Reports folder 52
- symptom database 142
- synchronization wrapper 292
- system performance tools 99
- System Service Tools (SST) 101, 106
- system services 294
- system tuning, semi-automatic 168
- system values 169
- System.out.println () 311
- System.out.println() 311

- system-level tools 28
 - for reactive use 93
- system-wide collection 42

T

- TCP/IP buffer size 210
- Technology Independent Machine Interface 353
- termination condition 290
- Test Manager 158
- think time 158
- thread 204
- thread interactions 127
- thread pool 279
- Thread pool maximum size 232
- thread priority 189
- thread state 173
- ThreadsPerChild 232
- ThreadsPerChild (directive) 205
- throughput 355
- throughput curve 234
- Time Slice End (TSE) 189
- time slice setting 188–189
- time watcher 342
- Tivoli Performance Viewer 18, 25, 117, 128, 130, 368
- TLS (Transport Layer Security) 204
- Toolbox 294–295
- Toolbox for Java JDBC driver 358
- Toolbox JDBC driver 357
- topology 2
 - Application 2
 - Database server 2
 - HTTP server 2
 - Java virtual machine 2
 - redundant 4
 - scaling 4
 - Web application server 2
- Total Size 339
- tperfviewer 130, 139
- Trace TCP/IP Application (TRCTCPAPP) command 151
- trading sites 348
- transaction 324
 - attributes 325
- transition data 182
- Transport Layer Security (TLS) 204
- trashing 191
- TRCTCPAPP command 151
- TreeMap 293
- TreeSet 293
- TSE (Time Slice End) 189
- tuning cache 202
- tuning process 23
- type 2 driver 295
- type 4 driver 295

U

- Unicode 303
- unsetEntityContext() 328
- useBean 316

V

- value object 328
- variable 292
- verboseGC 228
- vertical scaling 354
- virtualization 8–9
 - computational tasks 10
 - data-related activity 10
 - resource scavenging 10
- virtualized environment 9
- virtualized runtime 10
- Visual Explain 91

W

- wait state 173
- waiting request 232
- warm-up period 225
- Watched Threads folder 52
- WCBT (Work Control Block Table) 169
- Web application server 2
- Web browser 5
- Web server 2
 - performance 200
- Web service 8, 20
 - application-oriented architecture 1
 - service broker 8
 - service provider 8
 - service requester 8
- WebSphere
 - multiple instance support 354
 - scalability 354
 - stress testing applications 368
- WebSphere Application Server 2
 - security 19
- WebSphere Development Studio Client for iSeries 213
- WebSphere Development Studio Client for iSeries Advanced Edition 230
- WebSphere Dynacache messages 258
- WebSphere execution environment 17
- WebSphere performance tools 116
- WebSphere Resource Analyzer 117
- WebSphere Studio 25
- Whole Program Optimization (WPO) 13
- WLM (Workload Manager) 357
- Work Control Block Table (WCBT) 169
- Work with Active Jobs (WRKACTJOB) command 31
- Work with Disk Status display 29
- Work with PTF Groups (WRKPTFGRP) command 166
- Work with System Activity (WRKSYSACT) command 35
- Work with System Status (WRKSYSSTS) command 28
- Work with System Value (WRKSYSVAL) command 169
- workload 365
 - complexity 366
 - detail 365
 - type 364
- workload estimation tools 362
- Workload Estimator 362–363
- Workload Manager (WLM) 357
- workload patterns 347

- WPO (Whole Program Optimization) 13
- wrapper class 289
- WRITE_ONLY 308
- WRKACTJOB 25, 31
 - command 31
 - find 34
- WRKDSKSTS command 25, 29
- WRKOBJLCK command 100
- WRKPTFGRP (Work with PTF Groups) command 166
- WRKSHRPOOL command 175
- WRKSYSACT command 25, 35
- WRKSYSSTS command 25, 28, 182
- WRKSYSVAL (Work with System Value) command 169
- wsadmin 214

X

- XA transaction 281
- XML (Extensible Markup Language) 8



Maximum Performance with WebSphere Application Server V5.1 on iSeries

(0.5" spine)
0.475" <-> 0.873"
250 <-> 459 pages



Maximum Performance with WebSphere Application Server V5.1 on iSeries



Redbooks

The most complete book on WebSphere Application Server on iSeries performance tuning

End-to-end performance tuning methodology

Several good and practical examples

There may be several different reasons why you want to read this book. Perhaps you are currently experiencing a performance issue with an application. Or, you may be designing and developing a new application that you want to deploy on the IBM WebSphere Application Server running on the IBM *e*server iSeries server.

This IBM Redbook helps you to gain a solid understanding of the factors that can affect the performance of the Java and WebSphere applications running on the iSeries server. As a deployment platform, the iSeries architecture combined with Java and the WebSphere Application Server provides a robust execution environment, which is ideal for the mission-critical business applications and Web sites.

This book explains how to tune the iSeries server for IBM WebSphere Application Server. The book is divided into several logical tuning tasks, such as tuning the Java virtual machine. You can study each of these logical blocks and apply them independently from other tasks. But, to achieve optimum performance on the system, we recommend that you use the systematic approach that is shown in this book.

This book is written for the system administrators, application developers, and I/T consultants who are already familiar with WebSphere Application Server and work management topics for the iSeries server.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-6383-00

ISBN 0738491853