Performance Monitoring and Best Practices for WebSphere on z/OS

- Learn to build a monitoring and profiling environment
- Discover a comprehensive discussion of parameters
- Acquire knowledge about best practices

Alex Louwe Kooijmans
Tony J. Cox
Ruud Huisman
Magdalena Marsh
Koelli Mungee
Brian S Paskin
Martina Schmidt
Thomas Schulze
Javier Voos
Franck Yu
First Edition (April 2007)

This edition applies to WebSphere Application Server Version 6.02 and Version 6.1 for z/OS.

© Copyright International Business Machines Corporation 2007. 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 IBM Redbooks publication .......... xi
Become a published author ...................................... xiii
Comments welcome .............................................. xv

Chapter 1. Introduction ............................................. 1
1.1 The structure of the book ..................................... 2
1.2 How do you use this book ..................................... 2
1.3 Other sources of information ................................. 5

Part 1. Our environment .............................................. 7

Chapter 2. Our infrastructure ....................................... 9

Chapter 3. Our sample workload .................................... 19
3.1 IBM Trade V6.1 ................................................ 21
3.2 ITSO Trader V6.1 ............................................. 23
3.3 Business transaction scenarios ............................... 26
    3.3.1 IBM Trade V6.1 business transaction scenarios ... 27
    3.3.2 ITSO Trader V6.1 business transaction scenarios . 29
3.4 Workloads ........................................................ 30
    3.4.1 Workload scripts ....................................... 32

Chapter 4. Using workload drivers .................................. 33
4.1 Introduction ...................................................... 34
4.2 WebSphere Studio Workload Simulator (WSWS) ....... 34
4.3 Rational Performance Tester ................................. 37
4.4 JMeter ............................................................. 46
4.5 Others ............................................................... 52

Chapter 5. Setting up and using monitoring tools .................. 55
5.1 Performance problem determination ......................... 56
    5.1.1 Introduction ............................................. 56
    5.1.2 Problem symptoms and documentation for diagnosis 56
5.2 Monitoring and profiling tools ............................... 58
    5.2.1 SMF/RMF ............................................... 58
    5.2.2 Eclipse TPTP ........................................... 71
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.2.1 JVM storage allocation</td>
<td>193</td>
</tr>
<tr>
<td>8.2.2 Garbage Collection policy</td>
<td>195</td>
</tr>
<tr>
<td>8.2.3 Garbage Collection</td>
<td>196</td>
</tr>
<tr>
<td>8.3 Analyzing VerboseGC</td>
<td>197</td>
</tr>
<tr>
<td>8.3.1 Introduction</td>
<td>197</td>
</tr>
<tr>
<td>8.3.2 Our GC Analyzer tool</td>
<td>199</td>
</tr>
<tr>
<td>8.3.3 GC Analyzer charts</td>
<td>200</td>
</tr>
<tr>
<td>8.4 Just in Time (JIT) compilation</td>
<td>215</td>
</tr>
<tr>
<td>Chapter 9. z/OS subsystems</td>
<td>217</td>
</tr>
<tr>
<td>9.1 UNIX Systems Services</td>
<td>218</td>
</tr>
<tr>
<td>9.2 TCP/IP</td>
<td>220</td>
</tr>
<tr>
<td>9.3 Workload Manager (WLM)</td>
<td>222</td>
</tr>
<tr>
<td>9.3.1 WLM classification</td>
<td>222</td>
</tr>
<tr>
<td>9.3.2 Temporal affinity workload balancing</td>
<td>227</td>
</tr>
<tr>
<td>9.3.3 RMF Workload Activity Report</td>
<td>230</td>
</tr>
<tr>
<td>9.4 Resource Recovery Services (RRS)</td>
<td>233</td>
</tr>
<tr>
<td>9.5 Language Environment (LE)</td>
<td>234</td>
</tr>
<tr>
<td>9.6 Other considerations</td>
<td>235</td>
</tr>
<tr>
<td>9.6.1 GRS</td>
<td>235</td>
</tr>
<tr>
<td>9.6.2 Storage</td>
<td>236</td>
</tr>
<tr>
<td>Chapter 10. Connectors</td>
<td>237</td>
</tr>
<tr>
<td>10.1 WebSphere back-end connectivity</td>
<td>239</td>
</tr>
<tr>
<td>10.1.1 Common Connection pool properties</td>
<td>240</td>
</tr>
<tr>
<td>10.2 Java Database Connectivity (JDBC)</td>
<td>243</td>
</tr>
<tr>
<td>10.2.1 Data source tuning</td>
<td>244</td>
</tr>
<tr>
<td>10.2.2 WebSphere connection pool and DB2 threads tuning</td>
<td>248</td>
</tr>
<tr>
<td>10.2.3 Type 2 or Type 4 JDBC connectivity</td>
<td>249</td>
</tr>
<tr>
<td>10.3 Java Message Service (JMS) and WebSphere MQ</td>
<td>250</td>
</tr>
<tr>
<td>10.3.1 Message Listener Service</td>
<td>251</td>
</tr>
<tr>
<td>10.3.2 JMS connection factory settings</td>
<td>254</td>
</tr>
<tr>
<td>10.4 CICS Transaction Gateway</td>
<td>257</td>
</tr>
<tr>
<td>10.4.1 CICS Transaction Gateway for z/OS in local mode</td>
<td>258</td>
</tr>
<tr>
<td>10.4.2 CICS Transaction Gateway for z/OS in remote mode</td>
<td>260</td>
</tr>
<tr>
<td>10.5 IMS Connect</td>
<td>261</td>
</tr>
<tr>
<td>10.6 Using the Performance Monitoring Infrastructure (PMI)</td>
<td>263</td>
</tr>
<tr>
<td>10.7 References</td>
<td>266</td>
</tr>
<tr>
<td>Part 3. Best practices - application</td>
<td>267</td>
</tr>
<tr>
<td>Chapter 11. J2EE application best practices</td>
<td>269</td>
</tr>
<tr>
<td>11.1 Introduction to performance of J2EE applications</td>
<td>270</td>
</tr>
<tr>
<td>11.1.1 Terminology</td>
<td>270</td>
</tr>
</tbody>
</table>
11.1.2 Approach to improve performance ........................................ 271
11.1.3 Infrastructure tuning ....................................................... 272
11.2 Application profiling .......................................................... 273
11.3 J2EE overview ................................................................. 274
   11.3.1 JavaServer Pages (JSP) .................................................. 275
   11.3.2 Servlets ................................................................. 275
   11.3.3 Enterprise Java Beans (EJB) .......................................... 275
   11.3.4 Web services ........................................................... 276
   11.3.5 Other J2EE APIs ........................................................ 277
   11.3.6 J2EE and performance ................................................ 278
11.4 Application best practices .................................................. 278
   11.4.1 Java coding best practices ............................................. 278
   11.4.2 Web Container best practices ....................................... 284
   11.4.3 EJB Container best practices ....................................... 291

Chapter 12. Accessing EIS resources ............................................. 301
12.1 General hints ....................................................................... 302
12.2 Best practices for JDBC and SQLJ ......................................... 302
   12.2.1 Data type mapping ....................................................... 303
   12.2.2 Using static SQL ........................................................ 304
   12.2.3 AutoCommit ............................................................... 307
   12.2.4 Select and update ....................................................... 307
   12.2.5 Numbers ................................................................. 308
   12.2.6 Use DB2 built-in functions .......................................... 308
   12.2.7 Releasing resources ................................................... 308
   12.2.8 Connection pooling .................................................... 309
   12.2.9 getXXX() method ...................................................... 311
   12.2.10 Use of transactions ................................................... 312
   12.2.11 Statement caching ..................................................... 313
   12.2.12 Update batching ....................................................... 313
   12.2.13 Row prefetching ....................................................... 314
   12.2.14 Summary ............................................................... 315
   12.2.15 Additional best practices for SQLJ ............................... 316
12.3 Best practices for Java 2 Connector Architecture ......................... 320
   12.3.1 Re-use of objects ....................................................... 320
   12.3.2 Managed environment ................................................ 323
   12.3.3 Use of transactions ................................................... 323
   12.3.4 Connection pooling .................................................... 325
   12.3.5 Connection usage ..................................................... 325
   12.3.6 Lazy association ....................................................... 326
   12.3.7 Lazy enlistment ......................................................... 327
   12.3.8 Best practices for CICS Transaction Gateway .................. 328
   12.3.9 IMS Connect ........................................................... 329
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:

<table>
<thead>
<tr>
<th>Term</th>
<th>Term</th>
<th>Term</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-2-3®</td>
<td>IMS™</td>
<td>Redbooks (logo)™</td>
</tr>
<tr>
<td>AIX 5L™</td>
<td>Language Environment®</td>
<td>RACF®</td>
</tr>
<tr>
<td>AIX®</td>
<td>Lotus®</td>
<td>RMFTM</td>
</tr>
<tr>
<td>ClearCase®</td>
<td>Monitoring On Demand®</td>
<td>System z™</td>
</tr>
<tr>
<td>Cloudscape™</td>
<td>MVS™</td>
<td>Tivoli Enterprise™</td>
</tr>
<tr>
<td>CICS®</td>
<td>OMEGAMON®</td>
<td>Tivoli®</td>
</tr>
<tr>
<td>DB2®</td>
<td>OS/390®</td>
<td>WebSphere®</td>
</tr>
<tr>
<td>DRDA®</td>
<td>OS/400®</td>
<td>z/OS®</td>
</tr>
<tr>
<td>HiperSockets™</td>
<td>Rational®</td>
<td>zSeries®</td>
</tr>
<tr>
<td>IBM®</td>
<td>Redbooks™</td>
<td></td>
</tr>
</tbody>
</table>

The following terms are trademarks of other companies:

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates.

NOW, and the Network Appliance logo are trademarks or registered trademarks of Network Appliance, Inc. in the U.S. and other countries.

Enterprise JavaBeans, EJB, Java, Java Naming and Directory Interface, JavaBeans, JavaServer, JavaServer Pages, JDBC, JDK, JMX, JRE, JSP, JVM, J2EE, Solaris, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

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

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

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

Implementing a WebSphere® Application Server environment on z/OS® and deploying applications to it in a manner that delivers the best performance is not a trivial task. There are many parameters that can be influenced and applications can be written in many different ways.

This IBM® Redbooks™ publication will give you a structure you can use to set up an environment that is tuned to meet best performance and at the same can be monitored to catch eventual performance bottlenecks. We also pay attention to workload testing using a variety of tools and best practices in writing and deploying applications.

The team that wrote this IBM Redbooks publication

This IBM Redbooks publication was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Alex Louwe Kooijmans is a project leader with the International Technical Support Organization (ITSO) in Poughkeepsie, NY, and specializes in WebSphere, Java™, and SOA on System z™ with a focus on integration, security, high availability, and application development. Previously, he worked as a Client IT Architect in the Financial Services sector with IBM in The Netherlands, advising financial services companies on IT issues such as software and hardware strategy and on demand. Alex has also worked at the Technical Marketing Competence Center for IBM System z and Linux® in Boeblingen, Germany, providing support to customers starting up with Java and WebSphere on System z. From 1997 to 2002, Alex completed a previous assignment with the ITSO, managing various IBM Redbooks projects and delivering workshops around the world.

Tony J. Cox is a Certified IT Specialist working for IBM in the United Kingdom. He has 29 years of experience in a number of software related fields, and for the past three years has been performing services work for WebSphere Application Server and WebSphere MQ across a wide range of platforms. He holds a Masters degree in Information Engineering from the University of Southampton and his areas of expertise include messaging and transaction processing.
Ruud Huisman has worked in the IT industry for more than 25 years. Last year, he started at IBM in The Netherlands as an IT Specialist on WebSphere. Ruud has experience in many aspects of performance on z/OS. The last three years, he has worked on performance with WebSphere Application Server.

Magdalena Marsh is a Staff Software Engineer with WebSphere Application Server on z/OS Support in Poughkeepsie, New York. She has been supporting WebSphere on z/OS for over five years. She works with customers on various problems, including defects, customization, and performance issues. Magda holds a Bachelor of Science degree in Computer Science from the Binghamton University in New York and Master of Science degree in Computer Science from Rensselaer Polytechnic Institute in New York.

Koelli Mungee is a Staff Software Engineer in the WebSphere on z/OS support team based in Poughkeepsie, NY. She has five years of experience working with customers on WebSphere Application Server problems and specializes in security, connection management, and performance. Koelli holds a Bachelor of Science in Computer Engineering from the University at Buffalo, New York and a Master of Science in Computer Science from Rensselaer Polytechnic Institute, New York.

Brian S. Paskin is a Senior IT Specialist for IBM Software Services for WebSphere in the New England area for IBM USA. He has 13 years of experience with IBM, predominantly with WebSphere AS, WebSphere MQ, CICS®, and aggregated system design. He previously worked for IBM Germany and IBM Italy.

Martina Schmidt is an IBM internal student writing a diploma thesis about Java EE performance considerations on z/OS for a degree in Applied Computer Science from the University of Cooperative Education in Stuttgart (Germany).

Thomas Schulze is an IT Specialist working in System z pre-sales support in Germany. He has five years of experience in the System z field. He holds a degree in Computer Science from The University of Cooperative Education in Mannheim (Germany). His areas of expertise include WebSphere Application Server and z/OS systems programming.

Javier Voos works at Intel® Software and Services Group (SSG) as product manager for SOA solutions in Argentina. He is an IBM Certified Advanced System Administrator and IBM Certified SOA Solution Designer. He has worked at IBM Software Technical Support and SOA consulting for the Spanish South America region. He has twelve years of experience in the application development field. He has been working extensively with WebSphere Application Server and tools for the last six years, helping customers to design, implement, and tune J2EE™ applications. For the past five years, he has been working as project leader for a J2EE development framework based on patterns in the
Clinical Engineering R&D Center, Argentina. He holds a degree in Information Technology Engineering from the U.T.N. Córdoba University, Argentina. His areas of expertise include SOA, Java EE architecture design, WebSphere Application Server consulting services, and mainframe systems integration.

Franck Yu is a software IT Specialist at the Benchmark Center System z in Montpellier, France. He has 18 years of experience in software development, including software analysis, design, programming, and testing. He holds engineering degrees from Nice University, France and from Beijing University, China. His recent technical interests focus on Java performance test and code optimization.

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

Richard M Conway, Ella Buslovich, Robert Haimowitz
International Technical Support Organization, Poughkeepsie Center

Phil Wakelin and Dan McGinnes
IBM Hursley lab, United Kingdom

John Campbell
Distinguished Engineer, DB2® for z/OS Development, IBM Silicon Valley Laboratory

Ken Blackman
IMS™ - Advanced Technical Support (ATS), Americas

Jim Cunningham and Bob St. John
IBM Poughkeepsie lab, United States

Become a published author

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

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.
Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html
Comments welcome

Your comments are important to us!

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

- Use the online Contact us review IBM Redbooks form found at:
  
  ibm.com/redbooks

- Send your comments in an e-mail to:
  
  redbook@us.ibm.com

- Mail your comments to:
  
  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400
Introduction

The WebSphere Application Server environment and the J2EE programming model have become highly sophisticated and complex over the years. There are many adjustable parameters and artifacts in both the infrastructure and the application itself. From a performance perspective, “just” deploying a first application to WebSphere Application Server without significant preparation may lead to disappointment.

In this IBM Redbooks publication, we try to sketch a main line for preventing and detecting performance issues. There is a great deal of WebSphere performance-related information available in the public domain, also for WebSphere Application Server on z/OS specifically, and we are not repeating everything in this publication. However, by using this publication, you should have a structure to make sure you are approaching performance in the right away and also have a checklist of all the things to look for.
1.1 The structure of the book

The book is divided into three parts:

- Part 1, “Our environment” on page 7

  We built an environment in z/OS and used two applications to verify certain information: the ITSO Trader application and the new IBM Trade V6.1 application. We also created some workload scripts to verify the workings of the workload driver tools.

- Part 2, “Best practices - infrastructure” on page 143

  This part of the publication contains a lot of valuable performance best practices and background information for:
  - WebSphere Application Server and the IBM HTTP Server (Chapter 7, “WebSphere Application Server and HTTP Server” on page 159)
  - The Java Virtual Machine (Chapter 8, “Java Virtual Machine” on page 191)
  - z/OS subsystems used by WebSphere Application Server on z/OS (Chapter 9, “z/OS subsystems” on page 217)
  - Connectivity infrastructure used when connecting from WebSphere Application Server to DB2, CICS, IMS and WebSphere MQ (Chapter 10, “Connectors” on page 237)

- Part 3, “Best practices - application” on page 267

  Tuning the infrastructure is important, but is not the only success factor. The way an application is developed influences to a very large extent performance. This part contains the most important best practices in the application development area.

1.2 How do you use this book

You should use this book based on your role:

- If you are a first-time user of WebSphere Application Server on z/OS:
  - We recommend you set up an infrastructure on z/OS that is tuned to generic workloads. Use the recommended “starting” values for the adjustable parameters in:
    - Optionally, IBM HTTP Server (if it is part of your architecture)
      Refer to 7.2, “IBM HTTP Server and WebSphere Application Server plug-in” on page 161 for recommendations.
• WebSphere Application Server
  Refer to 7.3, “WebSphere Application Server” on page 170 for recommendations.

• Java Virtual Machine
  Especially take a look at heapsize settings and garbage collection policies. Refer to Chapter 8, “Java Virtual Machine” on page 191 for more details.

• z/OS subsystems.
  Especially take a look at Workload Manager (WLM). Refer to Chapter 9, “z/OS subsystems” on page 217 for more details on the z/OS subsystems used.

• Connectors.
  If your applications plan to access DB2, CICS, IMS, or WebSphere MQ, you should review the appropriate sections in Chapter 10, “Connectors” on page 237.

  – We also recommend looking at establishing an environment in which load can be generated to test the environment and its settings and monitoring can be performed to monitor the behavior of the environment under certain load conditions. This environment can be used in conjunction with the IBM Trade V6.1 benchmarking application and for any application that will be deployed to WebSphere Application Server.

  • For workload driver tools, look at Chapter 4, “Using workload drivers” on page 33.

  • For monitoring solutions, look at Chapter 5, “Setting up and using monitoring tools” on page 55.

  – To test the effectiveness of the settings in your environment, you may want to download the IBM Trade V6.1 application, deploy it, and run it under certain workload conditions. You can use a workload driver tool to drive various workloads and a monitoring solution to monitor the behavior of the environment and the application. For more details on IBM Trade V6.1, refer to 3.1, “IBM Trade V6.1” on page 21 for a functional overview and to Appendix A, “Installation of the Trade V6.1 application” on page 341 for instructions on installing the IBM Trade V6.1 application.

  **Note:** The IBM Trade V6.1 does not include back-end connections to CICS and IMS. If your focus is on those connections, you may want to use the ITSO Trader application, which is included in the additional material for this book. Refer to Appendix B, “Additional material” on page 395 for instructions on getting the ITSO Trader application.
Many parameters have a “starting” value that works best in most situations, but that may need to be adjusted to another value depending on your specific application or workload volume needs. When driving different sizes of workloads, adjust the parameters when necessary, but try to stay within the recommended “intervals”.

If you already have a WebSphere Application Server environment on z/OS, but are planning to deploy a new application to it:

– Revisit the parameters in your environment. They may have been OK for your existing applications, but may need to be adjusted for new future application deployments. Use the following information for this activity:

  • Optionally, IBM HTTP Server (if it is part of your architecture)
    
    Refer to 7.2, “IBM HTTP Server and WebSphere Application Server plug-in” on page 161 for recommendations.
  
  • WebSphere Application Server
    
    Refer to 7.3, “WebSphere Application Server” on page 170 for recommendations.
  
  • Java Virtual Machine
    
    Especially take a look at heapsize settings and garbage collection policies. Refer to Chapter 8, “Java Virtual Machine” on page 191 for more details.
  
  • z/OS subsystems
    
    Especially take a look at Workload Manager (WLM). Refer to Chapter 9, “z/OS subsystems” on page 217 for more details on the z/OS subsystems used.
  
  • Connectors
    
    If your applications plan to access DB2, CICS, IMS, or WebSphere MQ, you should review the appropriate sections in Chapter 10, “Connectors” on page 237.

  – Perform code reviews on the new application to make sure they do not include bad practices. It would be better to prevent the usage of bad practices in the application. Review Part 3, “Best practices - application” on page 267 for this purpose.

  – Make sure you have a working load testing environment, including a workload driver tool and one or more monitoring solutions.

    • For workload driver tools, look at Chapter 4, “Using workload drivers” on page 33.
    
    • For monitoring solutions, look at Chapter 5, “Setting up and using monitoring tools” on page 55.
– Perform load tests with the new application, making sure you do not run into bottlenecks. Use a workload driver tool and one or multiple monitoring solutions for this. You may need to use a “profiling” tool if you think performance issues are caused by the application itself.

1.3 Other sources of information

Besides this publication, there are numerous other sources of information. The WebSphere Infocenter is always a good place to start, but also other IBM Redbooks, Technotes, and white papers contain performance-related information. Refer to “Related publications” on page 399 for a summary of other sources of information.
Our environment

In this part of the publication, we will describe the environment we used for our work. This part is organized as follows:

- In Chapter 2, “Our infrastructure” on page 9, we explain the infrastructure we used for our work.

- We installed and configured the Trade V6.1 application for running our testcases and using the tools discussed. Chapter 3, “Our sample workload” on page 19 contains a good description of the Trade V6.1 application.

- When testing a new application with a certain amount of workload, it can be very important to discover whether there are certain bottlenecks in the resources used to run the application. In Chapter 4, “Using workload drivers” on page 33, we discuss a number of tools that can be used to perform workload testing.

- Once an application is deployed and the behavior is not as expected, tools may be required to be able to look inside the application while it is running.
There are a number of ways to look into the application and its runtime environment. Chapter 5, “Setting up and using monitoring tools” on page 55 describes a number of tools and techniques to monitor the environment, both from an application and a system perspective.
Our infrastructure

This chapter will focus on the logical and physical setup of our environment in a top-down approach. First, we will describe the architectural overview, and then narrow down to the level of technical definitions in the environment, such as the Service and Report Class definitions we used as initial settings.

While other chapters in this publication will describe the used components in greater technical detail, this chapter will primarily focus on how the components fit together.

**Note:** At the time this IBM Redbooks publication was written, WebSphere Application Server for z/OS Version 6.1 was only available as a beta version and was not an announced and supported product yet.

We decided to run our tests on both releases, WebSphere Application Server for z/OS Version 6.0.2 with the 1.4.2 JDK™ and Version 6.1 with the 1.5 JDK, each in the 31-bit version.

On comments on whether or not we used a specific product in combination with WebSphere Application Server for z/OS Version 6.1, please refer to the corresponding section in Chapter 4, “Using workload drivers” on page 33 and Chapter 5, “Setting up and using monitoring tools” on page 55.
In order to write a publication that covers most of the setups that can be seen out in the field, we decided to include a variety of connectors and back-end systems in our performance tuning environment.

An overview of the used infrastructure is shown in Figure 2-1 on page 11. It can be broken down into the following categories:

Workload drivers: Different tools that are used to generate Virtual Users (VU) that simulate users testing our sample applications.
- WebSphere Studio Workload Simulator for z/OS and OS/390®
- Rational Performance Tester Version 6.1.2
- JMeter Version 2.1.1

For more details on the above tools, see Chapter 4, “Using workload drivers” on page 33.

Application environment: Includes the WebSphere Application Server, the databases, and back-end transaction systems:
- WebSphere Application Server for z/OS Version 6.0.2 Fix Level 8
- WebSphere Application Server for z/OS Version 6.1
- Customer Information Control System (CICS) Transaction Server Version 3.1
- DB2 Universal Database for z/OS Version 8 (DB2 V8)
- Information Management System Version 9 (IMS)

Monitoring tools: A collection of tools to collect utilization information from the WebSphere Application Server environment and the applications running in it.
- RMF Performance Monitor (RMF™ PM)
- ECLIPSE Test & Performance Tools Platform
- Rational Performance Tester
- IBM Tivoli Performance Viewer
- IBM Tivoli Composite Application Manager (ITCAM)

For more details, see Chapter 5, “Setting up and using monitoring tools” on page 55.
As described earlier in this chapter, we used both WebSphere Application Server for z/OS Version 6.0.2.8 and a beta version of the Version 6.1 release.

One of the major differences in terms of used components between Version 6.0.2 and Version 6.1 is the different Java Development Kit (JDK). While the 6.0.2 uses the JDK 1.4.2, the WebSphere Application Server Version 6.1 uses the JDK 1.5, which is also referred to as SDK 5.0. This affected some of the load drivers and monitoring tools so that not all tests were run with both releases of WebSphere Application Server with all the different tools.

To simulate a production environment, we turned Global Security on for both releases.

**Note:** For more information about WebSphere Application Server Version 6.1, go to:

The application infrastructure also includes databases and back-end transaction systems. Although a variety of database and transaction systems can be used with WebSphere Application Server for z/OS, we choose to use the following systems that, in our opinion, resemble the most commonly used ones:

- CICS TS Version 3.1
- DB2 Version 8
- IMS Version 9
- Derby Version 10.1 (open source database used for application messaging)

DB2 is configured for data sharing and CICS for High Availability in a CICSPlex. While there are two instances of the external WebSphere MQ that we used for message queuing, these were not clustered. Figure 2-2 shows the LPAR placement of the subsystems.

**Note:** The scope of this IBM Redbooks publication is the optimization of the WebSphere Application Server for z/OS; therefore, the environment used is not optimized for end-to-end for performance.
We installed the XML C++ Parser 1.4 for z/OS (XML Parser) for the ECLIPSE monitoring tools and an Agent Controller for ECLIPSE and a different one for ITCAM. For details on prerequisites, refer to Chapter 4, “Using workload drivers” on page 33 for the workload drivers. For information about the monitoring tools, see Chapter 5, “Setting up and using monitoring tools” on page 55.

The WebSphere environment was set up as a Network Deployment installation with one cell that spans two server nodes. These nodes created a high availability cluster. In front of the nodes was a Web server that distributed incoming HTTP and HTTPS requests to both nodes on a “round robin” base. As mentioned before, we used an external WebSphere MQ for message queuing.

Figure 2-3 shows a detailed view on the WebSphere installation. The dotted squares show the WebSphere Application Server server nodes that form the cell. The squares at the bottom of each LPAR frame symbolize external components that must be installed in order to communicate to the various back ends.

The WebSphere application environment is an IBM System z 990-C24.

The two LPARs are defined with the following specifications:
LPAR “WTSC48”:
- 3 CPs (shared)
- 4 GB Storage
- z/OS 1.6

LPAR “WTSC52”:
- 2 CPs (shared)
- 1 zAAP (shared)
- 6 GB Storage
- z/OS 1.6

As a starting point for WebSphere Application Server, we used the settings displayed in Table 2-1.

Table 2-1  WLM settings for WebSphere system work

<table>
<thead>
<tr>
<th>Component</th>
<th>Service CLASS</th>
<th>Performance settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deployment Manager</td>
<td>SYSSTC</td>
<td>-</td>
</tr>
<tr>
<td>Agent</td>
<td>SYSSTC</td>
<td>-</td>
</tr>
<tr>
<td>Servants</td>
<td>SYSSTC</td>
<td>-</td>
</tr>
<tr>
<td>Daemon</td>
<td>VEL80</td>
<td>Execution Velocity of 80</td>
</tr>
</tbody>
</table>

For profiling and measurement, we decided to create CB entries for each node of the cell. They were set up as shown in Table 2-2.

Table 2-2  WLM settings for WebSphere enclaves

<table>
<thead>
<tr>
<th>Transaction Class</th>
<th>Service Class</th>
<th>Performance settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>WASPTENH</td>
<td>90% within 250 ms</td>
</tr>
<tr>
<td>Default</td>
<td>WASPTENM</td>
<td>90% within 750 ms</td>
</tr>
<tr>
<td>Medium</td>
<td>WASPTENM</td>
<td>90% within 750 ms</td>
</tr>
<tr>
<td>Low</td>
<td>WASPTENL</td>
<td>Execution Velocity of 10</td>
</tr>
</tbody>
</table>
The initial settings for Report Classes included one for each component of the WebSphere Application Server for each node. The back ends are covered in general by creating Report Classes for the whole installation, that is, both LPARs. Figure 2-4 shows a detailed overview of the Report Classes.

<table>
<thead>
<tr>
<th>Action</th>
<th>Type</th>
<th>Name</th>
<th>Start</th>
<th>Service</th>
<th>Report</th>
</tr>
</thead>
<tbody>
<tr>
<td>____</td>
<td>TN</td>
<td>PEAGNTA</td>
<td>___</td>
<td>SYSSTC</td>
<td>WASAGNAE</td>
</tr>
<tr>
<td>____</td>
<td>TN</td>
<td>PEAGNTB</td>
<td>___</td>
<td>SYSSTC</td>
<td>WASAGNBE</td>
</tr>
<tr>
<td>____</td>
<td>TN</td>
<td>PEDEMNA</td>
<td>___</td>
<td>VEL80</td>
<td>WASDMNAE</td>
</tr>
<tr>
<td>____</td>
<td>TN</td>
<td>PEDMG*</td>
<td>___</td>
<td>SYSSTC</td>
<td>WASDMGRE</td>
</tr>
<tr>
<td>____</td>
<td>TN</td>
<td>PESR01*</td>
<td>___</td>
<td>SYSSTC</td>
<td>WASSR1AE</td>
</tr>
<tr>
<td>____</td>
<td>TN</td>
<td>PESR02*</td>
<td>___</td>
<td>SYSSTC</td>
<td>WASSR2BE</td>
</tr>
<tr>
<td>____</td>
<td>TN</td>
<td>PFAGNTA</td>
<td>___</td>
<td>SYSSTC</td>
<td>WASAGNAF</td>
</tr>
<tr>
<td>____</td>
<td>TN</td>
<td>PFAGNTB</td>
<td>___</td>
<td>SYSSTC</td>
<td>WASAGNBF</td>
</tr>
<tr>
<td>____</td>
<td>TN</td>
<td>PFDEMNA</td>
<td>___</td>
<td>VEL80</td>
<td>WASDMNAF</td>
</tr>
<tr>
<td>____</td>
<td>TN</td>
<td>PFDMG*</td>
<td>___</td>
<td>SYSSTC</td>
<td>WASDMGRF</td>
</tr>
<tr>
<td>____</td>
<td>TN</td>
<td>PFSR01*</td>
<td>___</td>
<td>SYSSTC</td>
<td>WASSR1AF</td>
</tr>
<tr>
<td>____</td>
<td>TN</td>
<td>PFSR02*</td>
<td>___</td>
<td>SYSSTC</td>
<td>WASSR2BF</td>
</tr>
<tr>
<td>____</td>
<td>TN</td>
<td>D8I%DBM1</td>
<td>___</td>
<td>SYSSTC</td>
<td>DBWASPEF</td>
</tr>
<tr>
<td>____</td>
<td>TN</td>
<td>SCSCCE*</td>
<td>___</td>
<td>_______</td>
<td>CICSWASP</td>
</tr>
<tr>
<td>____</td>
<td>TN</td>
<td>ERWCTG</td>
<td>___</td>
<td>_______</td>
<td>CICSWASP</td>
</tr>
</tbody>
</table>

*Figure 2-4  Report Classes used in initial setting*
The above settings are more generic settings. When we started to run some tests, we realized that even though it is good to have separate Reporting Classes when looking at RMF records, the Report Class report is more limited than the Service Class report. Whenever RMF data is reviewed in this book, it is based on the Service Class specification shown in Figure 2-5. Each Service Class has the same definition shown in Figure 2-6.

**Tip:** If you have performance problems on a specific server, consider creating a separate WLM Service Class for that server. RMF Reports group the performance results by Service Class. Service Class reports provides more information than Report Class reports.

<table>
<thead>
<tr>
<th>Subsystem Type CB - WebSphere App Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qualifier</td>
</tr>
<tr>
<td>--------------</td>
</tr>
<tr>
<td># type</td>
</tr>
<tr>
<td>------------</td>
</tr>
<tr>
<td>1 SI</td>
</tr>
<tr>
<td>1 SI</td>
</tr>
<tr>
<td>1 SI</td>
</tr>
<tr>
<td>1 SI</td>
</tr>
<tr>
<td>1 SI</td>
</tr>
<tr>
<td>1 SI</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

*Service Class WASSR1AE - 6.0 PESR0* servers

Created by user MMATY on 2006/05/24 at 11:52:40
Base last updated by user MMATY on 2006/06/02 at 16:17:50

Base goal:
CPU Critical flag: NO

<table>
<thead>
<tr>
<th># Duration</th>
<th>Imp</th>
<th>Goal description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>90% complete within 00:00:00.750</td>
</tr>
</tbody>
</table>
In addition, we used the WebSphere settings shown in Table 2-3. If not stated otherwise these settings apply to both the Version 6.0.2 and the Version 6.1 releases.

**Table 2-3  General WebSphere settings**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global Security</td>
<td>On</td>
</tr>
<tr>
<td>WLM # Servants (min/max)</td>
<td>1/4</td>
</tr>
<tr>
<td>Workload profile</td>
<td>IOBOUND</td>
</tr>
<tr>
<td>JAVA2 security</td>
<td>Disabled</td>
</tr>
<tr>
<td>Application security</td>
<td>Enabled in WebSphere Application Server Version 6.1</td>
</tr>
</tbody>
</table>
Our sample workload

In this chapter, we describe the J2EE applications that we chose for our performance tests, as well as the business transaction scenarios that we defined to compose workloads, and finally the workloads that we used in our performance test.

Our selection criteria of the applications are based on their abilities in support of:

- Multi-tier architecture, in particular, the three-tier architecture: client tier, middle tier, and EIS tier.
- Multiple types of clients with different input channels, including the Web client using the HTTP protocol, the Web services client using the HTTP/SOAP protocol, and the EJB™ client using the RMI/IIOP protocol.
- Multiple back-end systems, including DB2, CICS, IMS, and MQ.
- Scalability, both vertical and horizontal, in a WebSphere Application Server clustering environment.
Also, the applications should demonstrate the use of typical J2EE components and technologies, such as servlets, EJB session beans, EJB entity beans and message driven beans, JDBC™ drivers, J2C connectors, and Web services. Figure 3-1 depicts the architecture of such application.

![Diagram of Application Architecture](image)

**Figure 3-1 Application architecture**

We realize that it is difficult to find a single application of a reasonable size that meets all our selection criteria. For this reason, we finally chose two applications instead of one that are complementary to each other in function: They are IBM Trade V6.1 and IBM ITSO Trader V6.1. They are both stock brokerage applications and support both the three-tier architecture and the three types of clients, so we can define similar business transaction scenarios and workload. One main difference between them is in the back-end supported systems: IBM Trade V6.1 only uses DB2 or an equivalent relational database system to store data while ITSO Trader works with CICS, IMS, and MQ.

In the following sections, we discuss in detail the two applications' architectures and the business scenarios that we can run and the workloads that we define for the performance tests.
3.1 IBM Trade V6.1

IBM Trade V6.1 is an end-to-end benchmark and performance test J2EE application for WebSphere Application Server. Figure 3-2 shows the topology of Trade V6.1.

![IBM Trade V6.1 topology](image)

Figure 3-2  IBM Trade V6.1 topology

Trade V6.1 main features include:

- Support of three-tier architecture
- Support of multiple types of clients, not only the three types of clients that we required: Web client, Web services client, and EJB client, but also JMS client.
- Support of back-end relational database systems, such as DB2.
- Support of messaging either using the WebSphere Application Server internal messaging provider, called the Service Integration Bus (SIB), or an external messaging provider, such as WebSphere MQ.
Use of J2EE components such as servlets, JavaServer™ Pages™ (JSPs), EJB session beans, EJB CMP entity beans, message driven beans, and Web services.

However, in our performance test, we do not cover all the functionalities of Trade V.61. Figure 3-3 shows the features and components that we used.

Trade V6.1 provides a common service interface that defines a central entry point to access the application from all input channels. This common interface is implemented by a stateless session bean that plays the role of a façade and that encapsulates the entity beans in which business logic is coded. As a result, the business logic implementation is isolated from the client code: changes in the implementation code would not lead to changes in client code as long as the prototypes of services in the common interface remain the same. The support of Web services offers even more flexibility and agility through loose coupling: the
common service interface can be extended, enhanced, even re-implemented by other different types of components than EJB session beans without impact on Web services clients.

Table 3-1 summarizes the main services defined in the common service interface.

Table 3-1  IBM Trade service interface

<table>
<thead>
<tr>
<th>Service name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>login(userId, password)</td>
<td>Log in a user using user ID and password.</td>
</tr>
<tr>
<td>getAccountData(userId)</td>
<td>Get the account data of the user(userId).</td>
</tr>
<tr>
<td>getHoldings(userId)</td>
<td>Get the list of all the user(userId)’s holdings.</td>
</tr>
<tr>
<td>getHolding(holdingId)</td>
<td>Get the holding (holdingId)’s data.</td>
</tr>
<tr>
<td>getQuote(stockSymbol)</td>
<td>Get the current quote of the stock(stockSymbol).</td>
</tr>
<tr>
<td>sell(userId, holdingId)</td>
<td>Sell the holding(holdingId) of the user(userId).</td>
</tr>
<tr>
<td>buy(userId, stockSymbol, quantity)</td>
<td>Sell a quantity of the stock(stockId) for the user(userId).</td>
</tr>
<tr>
<td>logout(userId)</td>
<td>Log out the user(userId).</td>
</tr>
</tbody>
</table>

3.2 ITSO Trader V6.1

The ITSO Trader V6.1 application is based on four existing J2EE applications:

- Trader CICS
- Trader IMS
- Trader MQ CICS
- Trader MQ IMS

They are all stock brokerage applications, developed during various IBM Redbooks publications projects over the past few years. *WebSphere for z/OS V6 Connectivity Handbook*, SG24-7064, is one the IBM Redbooks publications based on the ITSO Trader. We have merged these four applications into a single one and named this new application ITSO Trader V6.1. The following are the main changes that we have made in the new application:

- We used Rational® Application Developer V6 as the application development environment for Trader V6.1 instead of WebSphere Studio Application
Developer for the old Trader applications. One benefit of using Rational Application Developer for EJB development is that the tool creates an EJB client project and puts all EJB interfaces into this project, including remote and local home interfaces, business interfaces, and service interfaces. In this project, we also put all value classes referenced in the EJB interfaces. This EJB client project is shared both by the EJB implementation classes and the EJB clients, instead of having two copies, of which one has the EJB implementation classes and one has the EJB client code. This was inconvenient in the old Trader applications.

- We defined three session beans, one working with CICS, one with IMS, and one with MQ, and they are all stateless instead of stateful session beans, as used in the old Trader applications. Stateless session beans perform better than stateful session beans.

- We defined a common service interface to all types of clients, which is implemented by all the session beans.

- We used a single common servlet to serve Web clients working with CICS, IMS, and MQ. We did the same with the JSPs.

- We added support for Web services clients.
Figure 3-4 shows the architecture of the new ITSO trader application.

ITSO Trader V6.1 now supports:

- Three-tier architecture.
- Multiple types of clients: Web client, Web services client, and EJB client, but not JMS client.
- Back-end systems: CICS and IMS using a J2C connector.
- Messaging through WebSphere MQ using JMS, which in turn forwards messages to CICS through the CICS bridge or to IMS through the IMS bridge.
- Use of J2EE components, such as servlets, JavaServer Pages, EJB session beans, message driven beans, and Web services.

ITSO Trader V6.1 provides the services shown in Figure 3-2 to its clients.
Table 3-2  ITSO Trader service interface

<table>
<thead>
<tr>
<th>Service name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logon(userId, password)</td>
<td>Log on a user using userId and password.</td>
</tr>
<tr>
<td>getCompanies()</td>
<td>Get the list of companies.</td>
</tr>
<tr>
<td>getQuotes(userId, companyId)</td>
<td>Get the quotes of the company(companyId)'s stock as well as the user(userId)'s holding of that company.</td>
</tr>
<tr>
<td>sell(userId, companyId)</td>
<td>Sell the user(userId)'s holding of the company(companyId)'s stock.</td>
</tr>
<tr>
<td>buy(userId, companyId, quantity)</td>
<td>Buy the quantity of the company(companyId)'s stocks for the user(userId).</td>
</tr>
<tr>
<td>logoff(userId)</td>
<td>Log off the user(userId).</td>
</tr>
</tbody>
</table>

3.3 Business transaction scenarios

We identify three typical user profiles:

- Users who log in, look at their account, holdings, and stock quotes, and log out.
- Users who log in, check their holdings and sell some holdings, and log out.
- Users who log in, check stock quotes and buy some stocks, and log out.

Accordingly, we define three business transaction scenarios:

- **Browse** scenario
- **Sell** scenario
- **Buy** scenario

All three business transaction scenarios can be applied to both IBM Trade V6.1 and the ITSO Trader V6.1 application. However, the content of each scenario is slightly different for IBM Trade and for ITSO Trader.
3.3.1 IBM Trade V6.1 business transaction scenarios

Figure 3-5 through Figure 3-7 on page 28 show a use case diagram of each scenario for IBM Trade V6.1.

![Figure 3-5 IBM Trade browse scenario](image)

![Figure 3-6 IBM Trade sell scenario](image)
Figure 3-7  IBM Trade buy scenario
3.3.2 ITSO Trader V6.1 business transaction scenarios

Figure 3-8 through Figure 3-10 on page 30 show a use case diagram of each scenario of ITSO Trader V6.1.

Figure 3-8  ITSO Trader browse scenario

Figure 3-9  ITSO Trader sell scenario
3.4 Workloads

A **workload** is a mix of business transaction scenarios composed of a number of browse scenarios, sell scenarios, and buy scenarios. Table 3-3 shows a sample workload using the three business transaction scenarios.

### Table 3-3  A sample workload

<table>
<thead>
<tr>
<th>Scenario name</th>
<th>% of users running the scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Browse scenario</td>
<td>60%</td>
</tr>
<tr>
<td>Sell scenario</td>
<td>20%</td>
</tr>
<tr>
<td>Buy scenario</td>
<td>20%</td>
</tr>
</tbody>
</table>

For a stock brokerage application, there are many more users who just browse than those who sell or buy. The goal is to have a mix of scenarios that is as close as the real world situation. Normally, a scenario with read only operations is lighter than the sell and buy scenarios requiring updates. Therefore, the choice of the mix of scenarios has impact on performance numbers, such as response time and transaction rate.
Each workload is realized by a script for a specific workload simulation driver. Moreover, we need to define a workload per type of client. For example, Table 3-4 shows the workload scripts for the JMeter workload driver in our project.

**Table 3-4 Workload scripts for JMeter**

<table>
<thead>
<tr>
<th>Application</th>
<th>Client type</th>
<th>Path</th>
<th>Script file name</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM Trade</td>
<td>Browser/Web client - HTTP request</td>
<td>\Jmeter\scripts\trade</td>
<td>tradeWeb.jmx</td>
</tr>
<tr>
<td></td>
<td>EJB client - IIOP request</td>
<td>\Jmeter\scripts\trade</td>
<td>tradeService.jmx</td>
</tr>
<tr>
<td></td>
<td>Web Service client - SOAP/HTTP request</td>
<td>\Jmeter\scripts\trade</td>
<td>tradeService.jmx</td>
</tr>
<tr>
<td></td>
<td>Web - EJB - Web Service clients</td>
<td>\Jmeter\scripts\trade</td>
<td>trade.jmx</td>
</tr>
<tr>
<td>ITSO Trader</td>
<td>Browser/Web client using CICS</td>
<td>\Jmeter\scripts\trader</td>
<td>traderCICS.jmx</td>
</tr>
<tr>
<td></td>
<td>Browser/Web client using IMS</td>
<td>\Jmeter\scripts\trader</td>
<td>traderIMS.jmx</td>
</tr>
<tr>
<td></td>
<td>Browser/Web client using MQ to CICS</td>
<td>\Jmeter\scripts\trader</td>
<td>traderMQCICS.jmx</td>
</tr>
<tr>
<td></td>
<td>TXT file provided to change the company name in the requests</td>
<td>\Jmeter\scripts\trader</td>
<td>companies.dat</td>
</tr>
<tr>
<td></td>
<td>Browser/Web client using CICS - IMS - MQ</td>
<td>\Jmeter\scripts\trader</td>
<td>trader.jmx</td>
</tr>
<tr>
<td>ITSO Trader DB</td>
<td>Browser/Web client using JDBC</td>
<td>\Jmeter\scripts\traderDB</td>
<td>traderDB_JDBC.jmx</td>
</tr>
<tr>
<td></td>
<td>Browser/Web client using EJB CMP</td>
<td>\Jmeter\scripts\traderDB</td>
<td>traderDB_CMP.jmx</td>
</tr>
<tr>
<td></td>
<td>Browser/Web client using SQLJ</td>
<td>\Jmeter\scripts\traderDB</td>
<td>traderDB_SQLJ.jmx</td>
</tr>
</tbody>
</table>
In a real world situation, the scenarios used by the workload for one input channel usually differ from the ones used for other input channels. Web clients are more likely used by customers, Web services clients by business partners, while EJB clients are typically used in intranet applications. To simplify things, we used the same business scenarios for input channels.

### 3.4.1 Workload scripts

We have included a number of sample workload scripts for usage in JMeter in the additional material for this book. Refer to 4.4, “JMeter” on page 46 for more details on JMeter and to Appendix B, “Additional material” on page 395 for instructions on getting the additional material.
Chapter 4. Using workload drivers

In this chapter, we describe the various workload drivers, their architecture, supported protocols, and scalability, that we used to generate load tests against our applications. This chapter covers the following workload drivers:

- WebSphere Studio Workload Simulator (WSWS)
- Rational Performance Tester (RPT)
- JMeter
- Some other workload drivers
4.1 Introduction

Load testing an application is a very important step in optimizing the Web application server environment, in our case, WebSphere Application Server, the underlying infrastructure, and the application itself being tested. Load testing reveals the capabilities, limitations, and possible failure points of the environment. A load test encompasses all protocols and access methods used by the applications. The results of load testing, in conjunction with monitoring tools, yield a concise and overall view of the environment and its limitations.

There are numerous tools that are available that help with load testing. They range from open source projects to proprietary software. Each load testing software has different methods and protocols for testing, and displays the results in numerous different ways. Some of the load testing software can scale to help distribute load to the systems being tested, while others are limited in their scalability.

4.2 WebSphere Studio Workload Simulator (WSWS)

*WebSphere Studio Workload Simulator (WSWS)* is a scalable test tool that can generate substantial loads that stress Web servers, WebSphere Application Servers, and the infrastructure surrounding an application.

WSWS consists of two components, a *Controller* and an *Engine*. For high scalability, the Engine, which generates the load used during load testing, is installed on a z/OS server. The load generated by the Engine can be used to test any Web serving environment, and is not limited to Web servers on the z/OS platform only. WSWS supports multiple Engines.

Figure 4-1 shows a single Windows® workstation Controller and multiple z/OS workload Engines are able to drive work to any Web server.
The Controller runs in the Windows environment. The Graphical User Interface (GUI) is used to manage all aspects of the load testing process. Test scripts can be created and edited, simulation runs can be set up, executed, and monitored and test results can be analyzed without leaving the GUI environment.

Figure 4-2 shows WWS’s main GUI window.

![Figure 4-2 WebSphere Studio Workload Simulator's GUI](image)

Test scripts are automatically generated by the capture function as the user navigates through a Web session. The capture function, launched with a simple mouse click, records the session's Web traffic and turns it into a test script ready for immediate playback. The scripts can be modified to add complexities, such as weight distributions and looping, to mimic the actions of a group of real users.

WSWS allows for variable content, such as logon IDs, to be incorporated into a script to simulate real life activity. Further monitoring and analysis can be accomplished by defining sections of the test script as a transaction.

The playback function allows for a recorded script to be played back to test the environment. There are many parameters available that can be set to aid in the load test. Time and loop counters, various delays, and the number of virtual users can be set to simulate real life conditions.
After a test script has been defined and the simulation launched, the test can be monitored in real time during execution. WWSW performance monitoring can be viewed through the Controller’s GUI or the data can be accessed through a Web browser, allowing the tester to view the results from a remote location.

Figure 4-3 shows WWSW running in playback mode and showing certain performance information about the current test.

![WebSphere Studio Workload Simulator’s performance GUI](image)

In addition to the real-time monitoring and analysis tools, WWSW provides a graphing tool that can be used to create a more detailed and in-depth analysis of the environment being tested. WWSW can graph various performance characteristics against the number of simulated users or against time, such as response time, throughput of data, or page elements, and CPU or memory utilization. An in-depth analysis of test results is possible by changing the logging level. Detailed analysis can include the activity of each simulated client.
Figure 4-4 shows WSWS’s ability to map statistics from a load driving test that ran. Several tests can be grouped together and viewed on a single graph.

WSWS supports HTTP, the Secure Socket Layers (SSL) security protocol, FTP, SOAP, and SMTP, and handles capture and playback through a SOCKS firewall.

**Note:** For a more in-depth overview and configuration of WebSphere Studio Workload Simulator, please see: [http://www.redbooks.ibm.com/redbooks/pdfs/sg246073.pdf](http://www.redbooks.ibm.com/redbooks/pdfs/sg246073.pdf)

### 4.3 Rational Performance Tester

*Rational Performance Tester (RPT)* is a full load and monitoring suite that uses a distributed model to load test systems. RPT is not limited to Web server load testing, but can include specially written Java classes that can be used to load test other types of systems.
RPT is based on the IBM Rational Software Development Platform (Rational SDP), which in itself is built on Eclipse. RPT refers to the graphical user interface (GUI). RPT is used in conjunction with the IBM Rational Agent Controller (RAC), which is a non-graphical application and used in distributed environments to generate load. RPT is available for use on Linux and Windows platforms, and RAC is available for use on Linux, Windows, and z/OS platforms. In our tests, we used RPT and RAC on the Windows Platform.

**Note:** The IBM Rational Agent Controller must be at least at the same level as the Rational Performance Tester to ensure compatibility.

**Note:** An AIX® version of the IBM Rational Agent Controller that works with RPT is planned and will be available in the future.

Figure 4-5 shows the architecture of Rational Performance Tester. The RPT GUI runs on the Windows or Linux platform. A RAC runs on the RPT machine and communicates with other RACs running on Linux, Windows, or z/OS platforms. The scripts are compiled into Java classes and are sent to the other platforms that are running RAC. Any platform running RAC can then execute a test to the target system. RPT natively only support HTTP and HTTPS, but custom classes can be used to use other protocols, like RMI/IIOP or JMS.
RPT can be run on the Windows and Linux platforms. The RPT GUI provides all the tools necessary to create, edit, and replay test scripts. The RPT GUI includes real-time detailed reports with graphics that indicate the progress and results of a test. The test scripts are compiled into Java classes for efficiency. A test can be run without distributing the workload to other systems; however, due to system limitations, such as processor speed or memory, the RPT local installation may not be able to generate a sufficient load to fully load test an application.

In a case where a distributed workload is needed to load test a system, a RAC must be installed on the same system running RPT and a remote system running Linux, Windows, or z/OS. In a distributed load test, RPT will communicate with the locally installed RAC, which in turn will communicate with the remote RACs. The Java classes that were compiled under the RPT interface are transferred and saved on the remote systems using RAC. The load test is then initiated by running the Java classes and the results are passed back from the remote RACs to the local RAC and finally to the RPT, which saves the data locally. RAC has been specially designed for low memory and processor usage. As a result, limited hardware resources are able to generate very high user loads during load testing.

Figure 4-6 shows RPT running on Linux or Windows communicating with the local RAC. The local RAC then communicates with the remote RACs running on Linux, Windows, or z/OS. The remote RACs then execute the load test against the targeted systems, such as a Web server or WebSphere Application Server. The results are passed from the remote RACs to the local RAC and then back to the RPT which saves the data.

There is no programming necessary to create, modify, or execute a load test. A load test is a graphical illustration of the Web pages that will be visited during execution. A SOCKS proxy is used to record a test script by navigating, through a browser, the Web pages that the test should capture and should mimic the actual use of a user of the system. The captured Web pages can be viewed and modified through a browser-like window in the RPT test editor. This window
shows what the user would see when visiting a selected page, helping to make a more informed decisions about how to modify load tests prior to execution.

Figure 4-7 shows the RPT GUI. The GUI shows a test script that was recorded in the upper right panel and the matching HTML representation of the data captured in the lower right panel.

![The Rational Performance Tester GUI](image)

Server responses may vary according to what data the user enters. Often, these responses have a bearing on future information that will pass between client and server. RPT makes these responses accessible for re-use in tests involving subsequent server requests. This capability ensures that during a load test the virtual users will accurately simulate the use of the system in a real world situation.

Test data can be varied during a test using a feature in RPT called a *Data Pool*. A Data Pool allows identified fields in a test script to use the values stored in the Data Pool during a load test. There are no limits to the amount of Data Pools used in a test script and a Data Pool can contain multiple fields. A small amount of data can be entered into a Data Pool within the RPT GUI. In cases where large
amounts of data are needed, the Data Pool can be populated by importing a Comma Separated Value (CSV) file.

Figure 4-8 shows a Data Pool that contains the Trade User ID and Password used during sign-on. The information can be used during a load test to replace the entered values during script capture with one of the values in the Data Pool.

![Datapool](image)

*Figure 4-8  A Data Pool with User ID and Password fields*

RPT contains a Test Scheduler that allows for different characteristics of a load test to be changed or enhanced. This feature provides the ability to mirror the load that occurs on a running system. The scheduler features items such as run time, think time, statistic gathering, and overall user load.

The RPT Test Scheduler can generate IBM WebSphere Application Monitor (WSAM) and IBM Tivoli® Composite Application Manager (ITCAM) reports through the API provided in the products. At the end of the load test, RPT contacts the WSAM/ITCAM Managing Server and creates a report for the groups specified in the RPT Test Scheduler.
Figure 4-9 Shows the section of the RPT Test Scheduler that allows RPT to contact WSAM/ITCAM and create a report based on a load test that has completed. The format of the Managing Server URL is http://<host>:=<port>/am/webapi for ITCAM. The Group Names field should contain the names of the groups that contain the servers on which the load test was performed. Group Names are separated by spaces.

![WebSphere Studio Application Monitor](image)

**Figure 4-9** The connection to WSAM/ITCAM in the test scheduler in Rational Performance Tester

Within WSAM/ITCAM a user must be created called *bmuser*. This user will be used to create the reports. The report can then be viewed through the Managing Server under the Performance Analysis menu and the Saved Reports item.

Figure 4-10 shows the reports that were generated in ITCAM after each RPT load test. The reports show detailed throughput for each server, and breaks down the data into several different charts and specific performance statistics based on applications and transactions. More information is available in 5.2.4, “IBM Tivoli Composite Application Manager” on page 109.

![Reports](image)

**Figure 4-10** ITCAM saved reports from an RPT load test
Figure 4-11 shows an ITCAM report based on a RPT load test. The pie chart represents the percentage of throughput of each application or transaction. Specific performance numbers are listed below the chart. More in-depth information can be displayed for each application or transaction to give more precise information about each aspect about the application or transaction.

The RPT Test Scheduler can be configured to include diverse user groups that have different characteristics. Each user group can be defined by a percentage or physical number of users. The user groups can specify on which remote servers to run the load test. Each user group can run on one or multiple remote servers, provided RAC is installed, and can better simulate an actual load. User groups can contain the same or different test scripts, loop counters, delays, and random selectors.
Figure 4-12 shows a RPT Test Scheduler that contains three separate user groups. Each user group is set up with different characteristics. The first group, Stock Browsers, will contain sixty percent of the users and will loop through the load test ten times. The second and third groups, Stock Buyers and Stock Sellers, each contain twenty percent of the users and will loop through their load test five times. Each user group is executing a different load test script, and the Stock Buyers group is scheduled to execute on several different remote systems that are running RAC.

Figure 4-12  RPT Test Scheduler shows the user groups, scripts, and characteristics of a load test

Custom Java classes can be included in a load test. These Java classes permit more flexibility for custom load tests. Java classes may be written to create special reports, communicate to another system using a protocol that is not supported by RPT, or to handle special logic that is needed for a load test. RPT includes some example code to help create custom Java classes more quickly.

IBM Rational ClearCase® LT is included with RPT. This permits groups of testers to write test scripts in parallel and to version their test scripts.

RPT generates performance and throughput reports in real time, enabling detection of performance problems at any time during a load test run. These reports provide multiple filtering and configuration options that can be set before, during, and after a test run. Additional reports are available at the conclusion of the test run to perform deeper analysis on items, such as response time percentile distributions. Custom reports can be created. A custom report can be tailored to the need of the tester. Data from diverse sources can be combined on
one report to display information important to the tester. Reports can include bar, line, or pie charts.

Figure 4-13 shows a line graph detailing the response time over physical time during a load test for each Web page. The graph includes all elements that were identified in the test script and assigns them a different symbol and color. The Performance Summary section shows the physical statistics for each element.

![Response vs. Time Detail](image)

**Figure 4-13  A Rational Performance Tester standard report**

In some cases, the problems with performance may be due to hardware problems rather than software bottlenecks. RPT can collect and display multiple server resource statistics, thereby exposing bottlenecks responsible for poor performance.

**Note:** At the time of writing, Rational Performance Tester did not support the SOAP protocol. A plug-in was being developed to handle this protocol.
4.4 JMeter

Apache JMeter (JMeter) is an open source Java based load driver that can be used in a distributed environment to load test systems. JMeter includes a large number of supported protocols, monitoring reports, and the ability to include Java plug-ins to expand functionality.

JMeter is written in Java and uses the Swing component of Java to create its Graphical User Interface (GUI). JMeter also includes a server component that is used in distributed environments to create heavier load. JMeter Version 1.9 works in most environments that contains a Java Runtime Environment (JRE™) Version 1.3 or above. The JMeter GUI requires a display that is possible to display graphics to create and modify test scripts. The JMeter GUI is not needed to perform load testing. The JMeter server component is non-graphical.
Figure 4-14 shows the architecture of Apache JMeter in a distributed environment. JMeter can run on most platforms with an installed JRE. A display is needed to create and modify test scripts, but is not required to run load tests. JMeter communicates with JMeter Servers on various other platforms that support a JRE to run a load test. The JMeter Servers can communicate with a wide variety of back-end systems, ranging from Web servers to message queuing (MQ) managers using Java Messaging Service (JMS). JMeter Version 2.1 supports HTTP, HTTPS, FTP, SOAP, JMS, JNDI, JDBC, and LDAP. Additional Java classes can be incorporated into JMeter to extend it to other protocols.

The JMeter GUI provides all the necessary tools to create, modify, and to perform load tests. The GUI includes some real-time monitoring tools that can be used to display the details of a load test. JMeter saves its test scripts in an XML format that can be modified or even created without the use of the JMeter GUI. Load tests can be run from the JMeter GUI or from the command line. Load tests running from a single system may not be able to generate enough load due to system limitations. In this event, JMeter can be used in a distributed manner and distributed the load testing to other systems that are running the JMeter Server.

Running JMeter in a distributed environment requires no other software to be running on the JMeter system. JMeter communicates with its remote JMeter Servers via Java’s Remote Method Invocation (RMI) protocol. On the remote systems, the JMeter Servers use Java’s RMI Registry to facilitate the
communication between JMeter and the JMeter Servers. JMeter passes the test script to the remote JMeter Servers to perform the load test. The data from the load test is passed back to the JMeter client where the data can be stored locally.

**Note:** Configuring JMeter and JMeter Servers requires some manual editing of configuration files and on certain systems requires some Java commands to be executed before starting the JMeter Server. More information about configuring JMeter for distributed processing can be found in the *JMeter User Guide:*

http://jakarta.apache.org/jmeter/usermanual/remote-test.html

Figure 4-15 shows the overall architecture for Apache JMeter. JMeter contacts the Java RMI Registry and the RMI Registry calls the JMeter Server. The JMeter Server then executes the load test against the targeted systems, such as a Web server, database, or WebSphere Application Server. The results are returned from the JMeter Server to JMeter using RMI.

![Figure 4-15 Apache JMeter architecture in a distributed environment](image)

There is no programming necessary to create, modify, or execute a load test. A load test is not limited to Web pages and can incorporate any supported protocols. For Web pages, JMeter provides an HTTP proxy that is used to navigate through Web pages and capture the user interaction. The navigation of the Web pages should imitate a real user of the system. For all protocols, JMeter provides *configuration elements* that can be configured to interact with other systems. All aspects of configuration elements can be modified to suit the needs of the load test.
Figure 4-16 shows the JMeter GUI. The left hand panel is a tree representing a load test. The right hand panel is the result of a captured HTTP request. The HTTP request can be modified to suit the need of the load test, including the parameters that are passed.

![Figure 4-16 The representation of an HTTP page in Apache JMeter](image)

Server responses may vary according to what data the user enters. Often, these responses have a bearing on future information that will pass between client and server. JMeter makes these responses accessible for re-use in tests involving subsequent server request. This capability ensures that during a load test the virtual users will accurately simulate the use of the system in a real world situation.

Test data can be varied during a load test using a feature in JMeter called User Parameters. User Parameters allow parameters in a Configuration Element to use the values stored in the User Parameters during a load test. There are no limits to the amount of User Parameters used and User Parameters can contain multiple fields. The necessary data can be entered into a User Parameters
JMeter GUI directly. For large amounts of data, the data within the User Parameters can be configured to read an external text file.

Figure 4-17 shows User Parameters representing the sign-on user ID and password. The data can be used during a load test to replace the entered values during script capture with one of the values in the User Parameters.

![Figure 4-17  JMeter User Parameters with the variable user ID and password fields](image)

JMeter incorporates different types of configuration elements that can be used to control a load test. **Pre and Post Processors** can be added to modify the data before and after a request. **Timers** can be added to create random and uniform test flow based on throughput or physical time. **Thread Groups** control the number of users that will be used for the specific section of the test script. Thread Groups permit scheduling of the load test run. JMeter also contains numerous logic controllers to control the flow of the test script.
Custom Java classes can be included in a load test. These Java classes permit more flexibility for custom load tests. Java classes may be written to create special reports, communicate to another system using a protocol that is not supported by JMeter, or to handle special logic that is needed for a load test.

JMeter generates performance and throughput reports in real time, enabling detection of performance problems at any time during a load test run. The reports that are to be used for load testing analysis must be included in the test script prior to the load test being run. The data from reports can be saved with varying configurable information. The graphs that are produced can be saved for future reference.

Figure 4-18 shows a report that is generated during a load test. The data in the report is updated in real time and gives valuable information about throughput, which Web pages are taking the most time, and any errors that have occurred during the load test.

![Aggregate Graph](image)

*Figure 4-18  A JMeter report showing data on a specific load test*
Figure 4-19 shows a bar graph representing the data from Figure 4-18 on page 51. The bar graph can be changed to represent several other statistics, and can be saved in a graphics format or the data can be exported as a Comma Separate Values (CSV) file.

![Bar graph showing test results in a graph](image)

**Figure 4-19** A JMeter report showing test results in a graph

**Note:** More information about Apache JMeter, including setup and configuration, is available on the official Apache JMeter Web site: [http://jakarta.apache.org/jmeter/](http://jakarta.apache.org/jmeter/)

### 4.5 Others

The tools we are using to drive load in our tests only comprise a minute set of load and performance test tools that are available on the market. The tools range from open source that is available on numerous platforms to vendor supplied software. Each one has different attributes, and depending on exactly what is to be tested and which protocols are needed, some tools are better than others. Some come with a feature rich GUI and others are command line driven.
**Note:** Wikipedia.org has an extensive list of open source and commercial load and performance testing tools. This is not a complete list, but is a starting point to investigate the tools that are available:

http://en.wikipedia.org/wiki/Load_and_performance_test_tools
Setting up and using monitoring tools
5.1 Performance problem determination

In this chapter, we talk about performance problem determination and how diagnosis techniques and tools can help.

5.1.1 Introduction

Performance is an important consideration for enterprise applications. Problems with performance are also the hardest to debug. There are a large number of factors that come into play, and this makes it difficult to pinpoint the problem area.

There are many monitoring and profiling tools that can be used with WebSphere Application Server for z/OS that can help in narrowing down performance problems. We will discuss these tools in detail later in this chapter starting with 5.2.1, “SMF/RMF” on page 58. We will look at how each tool can be used and what benefits it can provide in terms of performance problem determination.

Please refer to Diagnosing Performance Problems with WebSphere Application Server on z/OS, WP100678, located at http://www.ibm.com/support/techdocs, which has an extended checklist and valuable information about diagnosing and eliminating performance problems.

This book contains a wealth of best practices and place to look for when experiencing performance issues. Part 2, “Best practices - infrastructure” on page 143 and Part 3, “Best practices - application” on page 267 are dedicated to these issues.

5.1.2 Problem symptoms and documentation for diagnosis

There are different types of performance problems. Here are some examples along with suggestions on documentation that can be collected to help diagnose the problem.

**High response time**

The average response time of the application may not meet your goals. If this is the case, you should try to isolate which particular application request or requests are experiencing the delay. You should try to determine whether the delay is in the application request processing or the server processing. If it is determined that it is in the application, you would need to know how much time is spent in each phase of the particular request. There are application profiling tools that can be used to break down the transaction to see how much time each method is taking and this can be reviewed with the application developer to address the response time problem.
If the delay is in getting the application requests picked up by the WebSphere Application Server servant, then the WebSphere Application Server WLM goals should also be reviewed using the RMF Workload Activity reports. The number of threads per servant and number of servants may also be a contributing factor depending on workload. If the number of users of the application drastically increased and the response time went up, it could be that the application needs more resources to handle the new workload.

In WebSphere Application Server for z/OS, the default action is to ABEND the servant address space when a timeout occurs for an application request. The ABEND code is EC3 with a reason code of the form 0413000x. The server automatically takes an SVCDUMP in this case and the dump can be analyzed to determine which thread timed out and what it was doing when the timeout occurred. It also can give information about how long the request was processing in the servant and how long it took to get to the servant from the controller. If it spent most of its time waiting in the WLM queue to be picked up by the servant, then WLM goals should be looked at. If it spent all the time being processed in the servant, then that gives details on what the application is doing and can be further investigated using application profiling tools. Either way, it gives a clear indication of where the time was spent.

**High CPU utilization**

The CPU utilization of the WebSphere Application Server address space may be high due to some kind of looping application code or other reason and this could slow down the rest of the work being processed by the server. System monitors and RMF information should be used to identify the source. Otherwise, profiling tools like Eclipse TPTP can pinpoint the problem to specific applications and their components. If this does not reveal the cause of the problem, an MVS™ console dump of the address space experiencing the high CPU activity should be taken for analysis. It is important to maximize the MVS systrace before taking the console dump.

**Note:** The MVS command to maximize the systrace is TRACE ST,999K.

Optimally, two MVS console dumps should be taken some time apart for comparison and this may highlight certain thread activity in both dumps that can confirm the reason behind the high CPU.

**JVM heap usage**

The JVM™ heap size allocated for the application server should be tuned and the heap usage should be continuously monitored. *Verbose Garbage Collection (GC)* tracing can be enabled for the application server without any significant overhead. It is a good practice to keep this on for monitoring purposes. The GC
overhead should not be more than 5%; if it does go above this amount, the application should be reviewed.

If the JVM heap runs out of memory, the verbose GC traces and JVM heap dumps should be analyzed. There are number of tools that can be used to analyze the JVM Heap, and these are listed in more detail in chapter 4, “Diagnostic tools for WebSphere for z/OS”, in *WebSphere for z/OS Problem Determination Means and Tools*, REDP-6880. Depending on the results of the analysis, the problem may or may not be application related.

### 5.2 Monitoring and profiling tools

There are a number of monitoring and application profiling tools available to use with WebSphere Application Server for z/OS. The rest of this chapter will discuss the installation and use of these tools and how they can be used for problem determination. Such tools are important to effectively narrow down and debug a performance problem.

#### 5.2.1 SMF/RMF

This part of the book gives an overview of the Resource Measurement Facility (RMF) and System Management Facility (SMF) capabilities of the z/OS operating system. It also describes the architecture and the installation of the monitoring tool we used, the RMF Performance Monitoring Java Technology Edition (RMF PM), and introduces the SMF Browser.

**RMF overview**

A traditional way of analyzing z/OS environments is using RMF reports. RMF reports mostly utilize RMF data written to the SMF data sets. Therefore, one can do also historical analysis. The RMF data is also available for online monitoring.

While it gives you an outside view of how the operating system sees resource usage, it cannot debug the application or provide end-to-end transaction response time. In terms of performance, it can be used to have a closer look at the Workload Manager (WLM) information like the Performance Index (PI) and if a Service Class reaches its defined goals.
A brief overview of RMF is given in Figure 5-1.

**Figure 5-1  Conceptual overview of the RMF workflow**

**RMF Performance Monitor**
We used the *RMF Performance Monitoring Java Technology Edition (RMF PM)*, as it is easy to install and use and does not require a deep insight in z/OS while providing the powerful RMF monitoring functions.

**Overview**
RMF enables you to monitor the performance of your z/OS host or Linux image (IBM System z or Intel) from a workstation. You can manage z/OS sysplexes and Linux images from a single point of control by monitoring the resources of the corresponding system.
On z/OS, RMF PM takes its input data from a single data server on one system in the sysplex, which gathers the data from the RMF Monitor III on each image. This function is called the Distributed Data Server (DDS). If you want to monitor several sysplexes, each one needs to have an active DDS. RMF PM supports the complete set of metrics provided by the RMF Monitor III gatherer. This information includes general performance data, performance data for jobs, performance data for systems running in goal mode, and workload-related performance data, such as:

- WLM workloads
- WLM Service Classes and periods
- WLM Report Classes

You have the flexibility to create unique views that monitor the performance of the system while displaying real-time and historical data as bar charts. Additionally, you can combine data from different resources. Once created, these views can be saved as PerfDesks that can be exported and reused by other RMF PM users. With PerfDesks, you create a set of DataViewS customized to your monitored systems. DataViewS sample performance data into one or more series displayed as bar charts. You can also save the data from the DataViewS into spreadsheet format, for further processing with spreadsheet applications.
Figure 5-2 shows the user interface of RMF PM. On the left side, you see the resource structure of the selected sysplex. On the right side, an open PerfDesk with an active DataView is displayed.

The workstation component for RMF PM is available for Microsoft® Windows and also for Linux. For setup guidance for the Linux version, check the RMF home page:


**Prerequisites**

While the workstation installation files include everything to set up the software, the host part has some prerequisites:

- Active RMF Monitor III.
- Active RMF Distributed Data Server (DDS).
- Active and configured TCP/IP.
- Active UNIX® System Services (Open Edition).
- For needed Program Temporary Fix (PTF), refer to the readme files delivered.

For additional prerequisites, refer to the RMF PM documentation.
**Additional RMF resources**


**Installing Distributed Data Server**

This section describes the installation of Distributed Data Server that is a requirement for the RMF PM.

The Distributed Data Server (DDS) is a single data server on one system in the sysplex. The DDS gathers data from Monitor III, which is distributed on all systems in the sysplex. If you want to monitor several systems in a sysplex, you must set up a DDS host session on that system in the sysplex with the highest RMF release. Each system of the sysplex must have an active and synchronized Monitor III gatherer.

If you want to monitor several sysplexes, each one needs to have an active DDS and an active and synchronized Monitor III gatherer.

RMF PM takes its input data from the DDS. You can also access the performance data from the DDS using the RMF data on demand in a Web browser.

**Considerations for z/OS UNIX level of security**

If the BPX.DAEMON FACILITY resource is defined, your system has z/OS UNIX security and can exercise more control over your superusers. Because DDS runs as a daemon, it must have access to the BPX.DAEMON facility. You need to define all programs loaded by GPMSEERVE to PROGRAM CONTROL. Example 5-1 shows the definitions for Security Server (RACF®) for the defined user ID GPMSEERVE, which is assigned to the started task GPMSEERVE.

*Example 5-1 TSO commands for Security Server (RACF)*

```plaintext
PERMIT BPX.DAEMON CLASS(FACILITY) ID(GPMSERVE) ACCESS(READ)
RDEFINE PROGRAM GPM* ADDMEM('SYS1.SERBLINK'\//NOPADCHK) UACC(READ)
RDEFINE PROGRAM ERB* ADDMEM('SYS1.SERBLINK'\//NOPADCHK) UACC(READ)
RDEFINE PROGRAM CEEBINIT ADDMEM('CEE.SCEERUN'\//NOPADCHK) UACC(READ)
RDEFINE PROGRAM IEEMB878 ADDMEM('SYS1.LINKLIB'\//NOPADCHK) UACC(READ)
SETROPTS WHEN(PROGRAM)REFRESH
```
**Customization**

On all systems that you want to monitor, you need to start the Monitor III gatherer with identical MINTIME and SYNC options.

Also, make sure that the following prerequisites are met on your z/OS host:

- UNIX System Services is configured.
- TCP/IP under UNIX System Services is configured and active.

RMF provides a default parmlib member GPMSRV00, shown in Example 5-2. You can tailor this according to your needs, but we recommend you use the standard GPMSRV00 member.

**Example 5-2  GPMSRV00 default parmlib member**

```plaintext
CACHESLOTS(4) /* Number of timestamps in CACHE */
DEBUG_LEVEL(0) /* No informational messages */
SERVERHOST(*) /* Don’t bind to specific IP-Address */
MAXSESSIONS_INET(5) /* MaxNo RMF PM clients */
SESSION_PORT(8801) /* TCP/IP port number RMF PM */
TIMEOUT(0) /* No timeout */
DM_PORT(8802) /* Port Number for DM requests */
DM_ACCEPTHOST(*) /* Accept from all IP-addresses */
MAXSESSIONS_HTTP(20) /* MaxNo of concurrent HTTP requests */
HTTP_PORT(8803) /* Port number for HTTP requests */
HTTP_ALLOW(*) /* Mask for hosts that are allowed */
HTTP_NOAUTH() /* No server can access without auth. */
```

Ensure that the ports used to access DDS are open for communication:

- RMF PM uses the SESSION_PORT (default 8801) and can optionally also use the HTTP_PORT (default 8803).
- RMF data on demand in a Web browser uses the HTTP_PORT (default 8803).
Example 5-3 shows how to start the DDS using the default GPMSERVE procedure of SYS1.PROCLIB. If no member parameter is specified, the default parmlib member GPMSRV00 is used.

Example 5-3  Starting the DDS

S GPMSERVE
IRR812I PROFILE GPMSERVE.* (G) IN THE STARTED CLASS WAS USED 292 TO START GPMSERVE WITH JOBNAME GPMSERVE.
$HASP100 GPMSERVE ON STCINRDR
IEF695I START GPMSERVE WITH JOBNAME GPMSERVE IS ASSIGNED TO USER GPMSERVE
, GROUP SYS1
$HASP373 GPMSERVE STARTED
IEF403I GPMSERVE - STARTED
IEE252I MEMBER GPMSRV00 FOUND IN SYS1.IBM.PARMLIB
GPM060I RMF DISTRIBUTED DATA SERVER READY FOR COMMANDS

**DDS host trace**

DDS allows you to set up a trace for further problem determination. You have to modify the GPMSERVE start procedure. Replace the SYSOUT and SYSPRINT statements either with data set definitions or sysout class definitions, as in Example 5-4.

Example 5-4  Modified GPMSERVE procedure

//GPMSERVE PROC MEMBER=00
//STEP1 EXEC PGM=GPMDDSRV,REGION=0M,TIME=1440,
// PARM='TRAP(ON)/&MEMBER'
//GPMINI DD DISP=SHR,DSN=SYS1.SERBPWSV(GPMINI)
//GPMHTC DD DISP=SHR,DSN=SYS1.SERBPWSV(GPMHTC)
//CEEDUMP DD DUMMY
//SYSPRINT DD DISP=(NEW,CATLG),UNIT=SYSDA,SPACE=(TRK,(1000,500),RLSE),
// DSN=SYS1.RMF.PTRACE
//SYSOUT DD DISP=(NEW,CATLG),UNIT=SYSDA,SPACE=(TRK,(1000,500),RLSE),
// DSN=SYS1.RMF.OTRACE
// PEND
Use the `modify` command with parameter TRACEON to enable the trace, as shown in Example 5-5. To stop the trace, use the TRACEOFF parameter.

**Example 5-5  Enable the DDS trace**

```
F GPMSERVE,TRACEON
GPM052I TRACE IS NOW ON
F GPMSERVE,TRACEOFF
GPM052I TRACE IS NOW OFF
```

The DDS trace produces a lot of output. Therefore, you should ensure that your trace data sets are big enough and that you only create the trace when it is really necessary.

For detailed information about setting up a RMF environment, see *Effective zSeries Performance Monitoring Using Resource Measurement Facility*, SG24-6645.

**Installation and setup of RMF PM**

This section will give you a brief introduction on how to install and set up RMF PM.

**Installation of RMF PM**

- We recommend that you check the RMF home page for the latest RMF PM version:


- Download the file for Windows and install it by running the file.

- To start RMF PM, select **Start** → **Programs** → **IBM RMF Performance Management** → **RMF PM**.
**Initial setup**

After you installed and started the application, the window in Figure 5-3 will prompt you to insert some system information.

![Figure 5-3  RMF PM Sysplex information](image)

For details on the information needed, you may need to ask your system administrator.

After this task happens, a new Sysplex view is created that shows you all LPARs included in this sysplex. You can access it by selecting the **Resources** tab on the left side of the window. Before you create DataView, you first need to connect the workstation to the host. You can do it by double clicking the sysplex symbol. RMF PM will then connect to the host and prompt you for your password.
To create a PerfDesk (shown in Figure 5-4):

- Return to the PerfDesk tab in the left side of the window.
- From the menu bar, select **File → New → PerfDesk...**
- Enter the name for the new PerfDesk.
- Press the **Save...** button.

![Workflow for creating a new PerfDesk for RMF PM](image)
To create a DataView, follow these steps (see also Figure 5-5):

- Press the right mouse button and select **New DataView**.
- Type in a name, choose the alignment of the bars, and press **Ok**.
- Select a metric you want to monitor.
- Optionally, you can select to define resources you want to see. Depending on the information to be monitored, the filtering will be done by jobs, WLM service classes, WLM report classes, volumes, and so on. You can make multiple selections by pressing the filter.
- Click **Add**.

![Figure 5-5  Workflow for creating a new DataView for RMF PM](image)

To start the monitoring, press the **Start** icon in the DataView. You can also start the monitoring for the whole PerfDesk by pressing the **Start** button on the bottom of the window.
To export or save the graph data, press the icon. This will open the export prompt (see Figure 5-6).

![Figure 5-6 Export dialog in RMF PM](image)

- Select the time frame you want to save.
- Select the data you want to export.

**Note:** By default, only one series is selected in the Maximum row. To select more, hold the control key and use the mouse to select multiple series.

- Press **Save**.... Choose the location to store the data and press **Save**.

**Note:** RMF PM allows you to export the data used to create the graphs to your local workstation. This can only be done into a *.wk1 file. This format is used by the Lotus® 1-2-3® program, but the data can be imported into Microsoft Excel® by simply dragging and dropping the file into an open Microsoft Excel window.
SMF - System Management Facilities - Record 120

System Management Facilities (SMF) can be enabled to collect and record system and job-related information about the WebSphere for z/OS system. This information can be used to bill users, report system reliability, analyze your configuration, schedule work, identify system resource usage, and perform other performance-related tasks that your organization may require.

SMF record type 120 is reserved for WebSphere Application Server for z/OS. There are two types of records that are produced:

- **Activity records** are gathered as each activity within a server is completed. An activity is a logical unit of business function. It can be a server or user-initiated transaction.
- **Interval records** consist of data gathered at installation-specified intervals and provide capacity planning and reliability information.

The following are the subtypes that can be collected:

- **Subtype 1** Server Activity record
- **Subtype 3** the Server Interval record
- **Subtype 5** J2EE Container Activity Record
- **Subtype 6** J2EE Container Interval Record
- **Subtype 7** WebContainer Activity record
- **Subtype 8** WebContainer Interval record

In order to enable and collect SMF 120 recording, you will need to:

1. In the administrator console, select **Server Infrastructure → Java and Process Management → Process Definition → Control → Environment Entries**.

   To collect activity records, set the following properties:
   - server_SMF_server_activity_enabled = 1 (or true)
   - server_SMF_container_activity_enabled = 1 (or true)

   To collect interval records, set the following properties:
   - server_SMF_server_interval_enabled = 1 (or true)
   - server_SMF_container_interval_enabled = 1 (or true)
   - server_SMF_interval_length = n,

   where n is the interval, in seconds, that the system will use to write records for a server instance. Set this value to 0 to use the default SMF recording interval.
2. Save and synchronize the changes.

3. Edit the SMFPRMxx parmlib member and update the SYS or SUBSYS(STC,...) statement to include the type 120 record, for example:

   ```
   SUBSYS(STC,EXITS(IEFU29,IEFACTRT),INTERVAL(SMF,SYNC),
   TYPE(0,30,70:79,88,89,120,245))
   ```

4. Use the SET command to indicate which SMF parmlib member the system should use. You must issue the SET command before you start WebSphere Application Server. If you issue the command after the application server has started, SMF type 120 records will not be collected.

5. Restart the application server and run performance tests for your application.

6. Issue the following command from the MVS console to switch SMF datasets:
   ```
   I SMF
   ```

7. Run the SMF Dump program (IFASMFDP) to create a sequential data set from the raw data. A sample JCL is shown in z/OS MVS System Management Facilities (SMF), SA22-7630.

8. Format the output with SMF Browser or other tooling.

   SMF Browser for WebSphere Application Server for z/OS V5 and V6 can be used to provide a simple report. The package includes the source code. It can be downloaded at:
   ```
   ```

   You can also write your own tools to retrieve and calculate the information that is important to you.

### 5.2.2 Eclipse TPTP

This section will provide you with a very brief overview of the open source Eclipse Test & Performance Tools Platform (TPTP) Project and explain its value as a performance tuning tool, followed by a guide on how to set up a profiling environment with TPTP for z/OS.

It does not cover detailed information about the Eclipse Software Development Kit (SDK) itself. To learn more on the Eclipse SDK, go to

   ```
   http://www.eclipse.org/documentation/main.html
   ```
TPTP overview
TPTP is an open source core project to the Eclipse Integrated Development Environment (IDE) that evolved from the Hyades project. It is used to profile an application. A profile is an analysis of a software during its runtime. The collected information can be used to optimize an application in terms of resource usage and performance and to identify problem points. In general, profiling can be divided into three areas:

- **Performance**
  This part of profiling measures (that is, counts) the number of class calls or method invocations. With this information, it is possible to identify areas of an application that should be optimized because they are frequently used.

- **Storage usage**
  Identifies the storage behavior of an application. Application developers can track the effectiveness of the storage management and usage of their application.

- **Parallelism**
  Means that you get an overview of the thread behavior. You get a graphical flow of the method invocation over time for each thread used in the application. This allows you to identify possible deadlocks. Figure 5-7 on page 73 shows an example of such a graphical flow.

TPTP supports both major approaches to application profiling:

- **Statistical analysis**
  This analysis is mostly done to discover if an application has a performance issue rather than to pinpoint its cause. A statistical analysis is done at full application speed and without the need for any modifications to the software. It is easy to do, as no code modification is necessary.

  When this type of analysis is used to measure the effect of changes in the software or environment, you need a predefined workload scenario so that the tests are repeatable.

  Statistical analysis can mostly be carried out without the need for in-depth Java or application knowledge.

- **Fractional analysis**
  A fractional analysis measures each major component of an application. To do this analysis, you have to include timing calls at the start and stop of each component. The profiling tool uses these information to calculate the time spent in each of the component.
The TPTP platform project fully exploits the Eclipse Business Intelligence and Reporting Tools capabilities (BIRT) to create detailed reports on different statistics, such as CPU time consumption, memory usage and class, or method invocations while using user defined thresholds.

Using TPTP will result in a slight overhead while running the application that you profile. This overhead will be calculated by TPTP because of the number of events processed and this prediction will be subtracted from the measured data before it is being presented to the user.

**Note:** In most cases, when experiencing bad performance in a Web application environment, it is not just an issue of the Web server or the related infrastructure.

The complex structure of a modern application with different layers and the use of third-party libraries often slows down overall system performance. To identify the exact source of this bottleneck, you need to profile the application.

For information about how to build Java applications for a z/OS environment, refer to *Java Stand-alone Applications on z/OS Volume I*, SG24-7177.

Figure 5-7 shows an example of the many windows available in TPTP.

![Figure 5-7 Execution flow of JAVA methods recorded by TPTP](image)
We used the statistical approach for all the three profiling areas for this book.

**TPTP architecture**

Within the TPTP framework, data is gathered from an application and collected within a server instance by an *Agent Controller*. The Agent Controller is communicating with the workstation on which TPTP is installed. For this task, the Agent Controller needs three ports open.

By default, these are set to the values below, but can be changed to any convenient value:

- Port “port“: 10002
- Port “secure port“: 10003
- Port “file port“: 10005

For information about how to download and to install the Agent Controller, please refer to “Eclipse help and information material” on page 76.

**Note:** In this book, you will also find references to the IBM Agent Controller. This Agent Controller is used by Rational Performance Tester (RPT) and cannot be used by Eclipse TPTP.

To make a clear distinction between these two, we will use the term Eclipse Agent Controller for further references on the Agent Controller needed to run TPTP.
An installation of TPTP consists of the following components (as shown in Figure 5-8):

- Eclipse SDK
- TPTP plug-in
- Agent Controller
- Application to be traced
- Optional BIRT plug-in

Through the use of filters, it is possible to profile a running application in a application serving environment like a Web server. These filters can be set whether to include or exclude the specific packages or classes in the tracing.
Our TPTP environment
For this book, we used the following Eclipse TPTP configuration:

- Eclipse SDK 3.2.0
- Eclipse Modeling Framework (EMF) 2.2.0
- XML Schema Infoset Model (XSD) 2.2.0
- Eclipse TPTP 4.2.0
- IBM JAVA Runtime Environment 1.5.0
- BIRT 2.1
  - Graphic Editor Framework (GEF) 3.2
  - Data Tools Platform (DTP) 0.9

Eclipse help and information material
When you are new to TPTP and Eclipse in general, you might get lost with finding the information you need because there is so much.

We found the use of the windowcasts extremely helpful, as they show in detail the use of the basic functions of every project. To access them, go to the documentation section of the Eclipse project site you want to learn more about.

In addition, there is a Eclipse Infocenter that combines all the information about the various Eclipse projects. It can be accessed using the following URL:

http://help.eclipse.org

Installation and setup of Eclipse TPTP profiling environment
This section will guide you through the installation and configuration process of an Eclipse TPTP environment on a Microsoft Windows system. It assumes that you will install the environment, including the BIRT report facility on a system without a Eclipse SDK on it.

Prerequisites
The workstation part of TPTP needs the following components installed:

- JRE or JDK 1.4 or higher
- Eclipse SDK 3.2.0 M6
- EMF 2.2.0 M6
- Eclipse Web Tools Platform (WTP) 1.5.0 M6 - (optional) required by profile on server feature
- BIRT 2.1.0 - (optional)
Chapter 5. Setting up and using monitoring tools

- GEF 3.2 M6 - (optional) required only if BIRT is installed
- DTP 0.9 M6 - (optional) required only if BIRT is installed

On the host side you need:
- JRE or JDK 1.4 or higher (This is the JDK needed to run the Agent Controller, not the one to be monitored.)
- XML C++ Parser for z/OS
- TCP/IP

In some forums, we found the information that the TPTP Agent Controller is not working with JDK 1.5 while others claim to get it installed and set up without problems. In our installation, the Agent Controller was working without any modifications or difficulties on the JDK 1.5 system.

**Installation of TPTP plug-in**
Follow these steps to set up the TPTP environment. This section assumes that you install the “BIRT full install” package on your system.

**Note:** We used the BIRT full installation package. This contains the Eclipse SDK, EMF, XSD, BIRT, and GEF. To find it, go to http://www.eclipse.org/birt and select the Download option in the menu and search for the complete install option. This makes the installation easier, as you do not need to spend a lot of time researching the needed components.

1. Download the BIRT full installation zip file (http://eclipse.org/birt) into a temp directory on your workstation. The name should be something like birt-report-designer-all-in-one-2.1RC1a.zip. Make sure that your system meets the requirements.

2. Unzip the BIRT full installation file into the C:\eclipse file. Do not change the path, as the TPTP component will be installed by default in an Eclipse directory that is created or used from the starting point you specify while unzipping it. After this step has finished, you have a ready to run Eclipse SDK installation that could be used to develop applications.

3. Download a current TPTP version from http://eclipse.org/tptp into a temporary directory and make sure to meet the requirements connected to it.

4. Unzip the TPTP installation package. With TPTP Version 4.2, you had to unzip the file directly onto the C:\ drive, because its path is set to use the C:\eclipse directory whether or nor the base Eclipse installation is in that directory.
The unzip tool will ask you if you want to replace the following files:

- epl-v10.html - Eclipse public license
- notice.html - Eclipse Foundation Software User Agreement

Replacing or keeping these files has no impact on the functionality of the TPTP installation (the contents of the files is the same). We chose to replace the “old” versions.

5. Validate the TPTP installation

   a. Start the Eclipse SDK by using C:\eclipse\eclipse.exe.

   b. If this is the first time you bring up the application, Eclipse will ask you to define a workspace where Java projects will be stored. Select a convenient path (Figure 5-9).

   ![Workspace Launcher]

   Figure 5-9  Define TPTP workspace directory

   c. Look for this shortcut in the menu bar: 

   Right now, you could connect the TPTP to an Agent Controller. If you do not have a running Agent Controller on your target systems, refer to “Installation of Agent Controller” on page 79. If you have it already installed, refer to “My first TPTP application profile” on page 84.
Installation of Agent Controller

The Agent Controller needs to be installed once per node in which applications should be monitored. To do so, follow these steps:

Note: For security reason, we recommend creating a user only for the Agent Controller. Otherwise, you may risk canceling a server process when stopping a profile by using the wrong way to cancel a profile session.

For more information about this task, refer to “My first TPTP application profile” on page 84.

1. Make sure that the following software is installed and configured on your system:
   a. JRE or JDK 1.4 or higher
   b. XML C++ Parser for z/OS and included in your LIBPATH
   c. TCP/IP
   d. Three free ports defined to communicate with the TPTP installation

2. Download the Agent Controller for z/OS from [http://www.eclipse.org/tptp](http://www.eclipse.org/tptp) for the correct version of TPTP.

3. Unzip the file on your hard drive and then create a .tar file from it.

4. Transfer the file to the host and untar it.

5. Set the following permissions:
   - `../AgentController/bin` to 775
   - `../AgentController/lib` to 775
   - `../AgentController/config` to 664
6. To set the parameters for the Agent Controller, run the 
../AgentController/bin/SetConfig.sh (see Example 5-6). The settings will be 
saved in ../AgentController/bin/serviceconfig.xml.

Example 5-6  Agent Controller SetConfig.sh script dialog

Specify the fully qualified path of "java" 
(e.g./usr/javal.4/jre/bin/java): 
Default="/Z16RD1/usr/lpp/java/J1.4/bin/java" (Press <ENTER> to accept 
the default value) 
New value>

If The Default is OK press enter

Network access mode (ALL=allow any host, LOCAL=allow only this host, 
CUSTOM=list of hosts): 
Default="LOCAL" (Press <ENTER> to accept the default value) 
New value>

Enter ALL as we wish to allow all clients to connect

Please enter the JBoss Application Server Home: 
Default="" (Press <ENTER> to accept the default value) 
New value>

We do not have JBoss installed, press enter

Please enter the JOnAS Application Server Home: 
Default="" (Press <ENTER> to accept the default value) 
New value>

We do not have JOnAS installed, press Enter

Note: If you do not want to use the standard ports for the TPTP Agent 
Controller, you have to change the parameters manually in 
../AgentController/bin/serviceconfig.xml. The values for the ports can be found 
in line 2.

To check if a port is already in use, issue the following command:
netstat –P <specified portnumber>

7. Include the *.so files in ../AgentController/lib/ in the JVM - LIBPATH.

8. Ensure that you have the appropriate permissions to change the program 
control file attribute.
9. Make the libraries program controlled:
   
   ```
   extattr +p <filename>.so
   ```

10. Return to normal user.

11. Start the Agent Controller by executing the RAStart.sh in 
    `../AgentController/bin`.

   If everything is set up properly, the Agent Controller should come up with a 
   message similar to what is shown in Example 5-7.

   **Example 5-7  Eclipse Agent Controller start message**

   ```
   SCHULZE @ SC48:/u/agent1/AgentController/bin>./RAStart.sh
   Starting Agent Controller
   RAServer started successfully
   ```

   If it does not work, then follow the error message shown on the USS shell or look 
   in the log file found in `../AgentController/config/servicelog.log`.

You now have a running Agent Controller that can communicate to the Eclipse 
TPTP installation on your workstation. The next step to enable profiling is to 
prepare the WebSphere Application Server Node you want to monitor.

**Configuring the application to be profiled**

TPTP is capable of connecting itself to an already running application at any 
given time as long as the application was started with the proper JVM arguments. 
To monitor an application that is running within the WebSphere Application 
Server, you have to start the WebSphere Application Server servant with an 
additional JVM property.

To do this task, go through the following steps:

1. Log on to the administrative console.
2. Select **Servers → Application servers → <your node>s → Java and 
   Process Management → Process Definition → Servant → Java Virtual 
   Machine.**
3. In the Generic JVM arguments field, enter:
   ```
   -XrunpiAgent:server=enabled
   ```
4. Apply the changes using the administrative console.
5. Restart the servant address spaces.
**Note:** If the servant will not start, search the log file for any entries that link to a piAgent or a *.so file. If it says that these cannot be found, then check the LIBPATH setting and make sure the AgentController *.so files are included in it.

**Note:** If you change settings in the Agent Controller, we found that it is the best way to:

- End the Agent Controller.
- Make the changes to the Agent Controller. (Be aware that if you use the SetConfig.sh script to change settings, the ports will be set to the default settings of 10002, 10003, and 10005, so you need to change them afterwards.)
- Bring down the WebSphere servant.
- Start the Agent Controller (if it says that the ports are already in use but you have not changed them, just retry to start the Agent Controller, and then it should work).
- Start the WebSphere servant.

**TPTP window overview**

This section shows you the major components of the TPTP window. Keep in mind that the actual appearance on your window may vary, as the different windows will rearrange themselves as you open and close different views.
Figure 5-10 shows you an overview of the TPTP window with an open execution flow in the actual data view.

The window can be divided in five major components:

1. **Menu bar and shortcuts**
   
   This is the normal look and feel for applications running in application windows (whether on Windows or Linux).

2. **Navigation view**
   
   This is the main navigation section for Eclipse TPTP that contains multiple tabs. The standard tab for working with TPTP is the Profile Monitor. Here you can start and stop a profile trace and access the various data types selected. The tree structure shows you **Selected Monitor** (which typically will be DefaultMonitor) → **connected host** → **instance of connected Agent Controller** → **profiled resource type**. The instance of an Agent Controller refers to “historic data”. Each time you connect to the same Agent Controller after closing and starting TPTP, there will be a new instance available.
3. Data view

This view shows the collected data in various ways. Each resource has special statistical and graphical ways to present the data. In Figure 5-10 on page 83, the graphical execution flow can be seen. You can open multiple data views that will be arranged as tabs at the top of this view.

4. Editor view

This view contains an editor that shows you for example your exported data.

5. Infobar

This infobar contains specific information in the left half view. When recording a profile, you can see the number of events reported. This is of special interest if a threshold is defined after a certain amount of events is recorded and the profile is stopped. The right part presents general information about the heap size of TPTP and its usage.

Note: For detailed information about the GUI, refer to the TPTP documentation or the Eclipse infocenter at:

http://help.eclipse.org/help30/index.jsp

My first TPTP application profile

Now that you have a running profiling environment, this section shows you how to create a profile of an application already running in a WebSphere environment. It assumes that you have already started the Eclipse SDK and defined a workspace that is convenient for you. (A screen cast showing these steps is available in the documentation section for Eclipse TPTP at http://eclipse.org/tptp.).

1. Open the drop-down menu hidden in the shortcut by pressing the triangle.
2. Select the **Profile**... entry. This will open the window shown in Figure 5-11.

![Profile window](image)

*Figure 5-11  Create new TPTP profiling to attach to a running Java process*

3. On the left side, select **Attach - Java Process**.
4. Press the **New** button on the upper left side.
5. This changes the right side to the window shown in Figure 5-12.

![Figure 5-12 Configure TPTP profiling to attach to a running Java process](image)

In the Default Host frame, two connections exist per default after the installation, which cannot be deleted.

6. Press the Add... button.

7. Insert the host name or IP address and the Agent Controller port (Figure 5-13). The port to enter is the “port” (“TPTP architecture” on page 74) defined. By default, this would be the 10002 port, in our case, we used the 12002 port.

![Figure 5-13 Add new host for TPTP profiling](image)

8. Press OK.

9. You can press Test Connection to test if the connection and the AgentController are working.

10. Select the created Host and change to the Agents tab.
11. If the Agent Controller is running and an application is running that was started with the XrunpiAgent attribute, then you should see an entry for an “unknown agent” (See Figure 5-14).

![Figure 5-14 TPTP Agent selection tab](image)

12. Mark the entry and press }.

Figure 5-14  TPTP Agent selection tab
13. On the Monitor tab, you can select the information and the details you want to monitor. Mark the folder that you want to change and press *Edit Options* (Figure 5-15).

*Figure 5-15  TPTP tab for selecting monitoring details*
– Java Profiling

Here you can create filter sets for your profiling session. These allow you to select specific Java packages and methods that should be included or excluded. In addition, you can set limits for the profiling of an application. As the file size of a trace grows very fast, this is a good way to make sure that your workstation can still handle the amount of data collected (see Figure 5-16).

![Figure 5-16  General options for TPTP Java Profiling](image)

– Basic Memory Analysis

Here you can specify if the data being collected should be collected for each instance.

– Execution Time Analysis

In this window, you can select the detail level of the CPU usage. If the “Show execution flow graphical details” option is selected, TPTP will produce a graph similar to the one shown in Figure 5-7 on page 73. This allows the optical identification of loops or long waiting periods for threads and methods.

– Method Code Coverage

The code coverage function collects information about the number of calls on Java class or method level.
– Probe Insertion

If this option is selected, TPTP can use self-written code fragments to collect specific data.

14. On the tab, you can leave the defaults.

15. The tab allows you to change the files where the profile data is saved. For the first profile, you can use the defaults.

16. Press . This will open an Eclipse window where you can see the attached AgentControllers in the Profiling Monitor tab (see Figure 5-17). It is possible to attach a single TPTP installation in parallel to multiple Agent Controllers and it can profile multiple applications at the same time. Right now, the workstation is connected to the Agent Controller, but has not started profiling the application.

Figure 5-17  TPTP Profiling Monitor section

17. To start the profiling, press in the Profiling Monitor. In the infobar at the bottom of the window, you can see some collection statistics.

Figure 5-18  Infobar for profiling data collected

18. To stop the profiling, press .
Note: If you try to stop the profiling with the button, the window in Figure 5-19 will ask you to terminate the “unknown process” that is actually the servant region running on the host.

Depending on the permissions that are connected to the user ID that started the Agent Controller, you risk losing connection to the server. Actually, we have not seen the WebSphere Application Server freeze due to a wrong stop. What we saw, however, was that we lost the Agent Controller connection to the application, which resulted in a restart of the Agent Controller itself and the servant region we wanted to monitor.

![Profiling Message]

Figure 5-19 Warning when trying to stop Agent with stop button

19. The collected data can be accessed by double-clicking the information you want to see in the Profiling Monitor. You can also right-click the data and choose the Open With option (see Figure 5-20).

![Profiling Monitor]

Figure 5-20 Choose data view for profile data
The following data views are possible:

- Basic Memory Analysis
  - Memory Statistics
  - Object Reference
- Execution Time Analysis
  - Execution Flow
  - Execution Statistics
  - UML Interactions (on Class, Object and Thread level)
- Method Code Coverage
  - Coverage Statistics

If you want to detach from WebSphere, right-click the process ID and select **Detach from Agent**. If you want to re-connect to it later, just right-click the process ID and select **Attach to Agent**.

After the data is collected, you can create a BIRT report out of it. For information about how to do this task, please refer to the Eclipse BIRT documentation.

### 5.2.3 Performance Monitoring Infrastructure (PMI)

The *Performance Monitoring Infrastructure (PMI)* is the core monitoring infrastructure for WebSphere Application Server. The performance data provided by WebSphere PMI helps to monitor and tune the application server performance. PMI data is the major source of input to *Tivoli Performance Viewer (TPV)* and *IBM Tivoli Composite Application Manager (ITCAM)*.

When tuning WebSphere Application Server for optimal performance, or fixing a poorly performing Java 2 Platform, Enterprise Edition (J2EE) application, it is important to understand how the various run time and application resources are behaving from a performance perspective. PMI provides a comprehensive set of data that explains the run time and application resource behavior. For example, PMI provides database connection pool size, servlet response time, Enterprise JavaBeans™ (EJB) method response time, Java virtual machine (JVM) garbage collection time, CPU usage, and so on. This data can be used to understand the run time resource utilization patterns of the thread pool, connection pool, and so on, and the performance characteristics of the application components like servlets, JavaServer Pages (JSP™), and enterprise beans.
Using PMI data, the performance bottlenecks in the application server can be identified and fixed. For example, one of the PMI statistics in the Java DataBase Connectivity (JDBC) connection pool is the number of statements discarded from prepared statement cache. This statistic can be used to adjust the prepared statement cache size in order to minimize the discards and to improve the database query performance.

PMI data can also be used to monitor the health of the application server. Some of the health indicators are CPU usage, servlet response time, and JDBC query time. Performance management tools like Tivoli Monitoring for Web Infrastructure and other third-party tools can monitor the PMI data and generate alerts based on some predefined thresholds.

**Enabling PMI data collection**

Performance Monitoring Infrastructure (PMI) needs to be enabled (by default, PMI is enabled) before collecting any performance data. PMI should be enabled before the server starts. If PMI is enabled after the server is started, the server needs to be restarted to start the PMI.

When PMI service is enabled, the monitoring of individual components can be enabled or disabled dynamically. PMI provides four predefined statistic sets that can be used to enable a set of statistics. The following list provides details about the statistic sets. If the predefined statistic sets do not meet your monitoring requirement, the Custom option can be used to selectively enable or disable individual statistics. The statistic sets are:

- **None**
  All statistics are disabled.

- **Basic**
  Statistics specified in J2EE 1.4, as well as top statistics like CPU usage and live HTTP sessions, are enabled. This set provides basic performance data about run time and application components. This is the default setting.

- **Extended**
  The basic set, plus key statistics from various WebSphere Application Server components, such as WLM and Dynamic caching, are enabled. This set provides detailed performance data about various run time and application components.

- **All**
  All statistics are enabled.

- **Custom**
  Fine-grained control to enable/disable statistics individually.
To enable PMI from the Administrative Console, select **Monitoring and Tuning → Performance Monitoring Infrastructure (PMI) → <server name>**.

The default settings are shown in Figure 5-21.

The custom settings are shown in Figure 5-22 on page 95.

![Figure 5-21 Default PMI settings for a server](image-url)
Enabling the Java Virtual Machine profiler data

Use the Java Virtual Machine Tool Interface (JVMTI) to collect data about garbage collection and thread states from the JVM that runs the application server. When JVMTI is enabled, a collection of JVMTI-specific statistics can be enabled through the predefined PMI All statistic set or through the use of a PMI Custom statistic set.

To enable JVMTI data reporting for each individual application server, select Servers → Application Servers → Java and Process Management → Process Definition → Servant → Java virtual machine.
Type `-agentlib:pmiJvmtiProfiler` in the Generic JVM arguments field. This is shown in Figure 5-23.

**IBM Tivoli Performance Viewer (TPV)**

This section describes the *Tivoli Performance Viewer (TPV)* tool that can be used to monitor WebSphere Application Server for z/OS. In WebSphere Application Server Version 4, TPV was named the *Resource Analyzer*. In WebSphere Application Server Version 5, TPV was a stand-alone Java application. Starting with WebSphere Application Server for z/OS V6.0, TPV is embedded in the administrative console and does not require any extra setup.
The WebSphere InfoCenter has detailed information about TPV and parts of this chapter are extracted directly from there.

**TPV overview**

TPV is run from the administrative console to monitor the overall server performance. There are two topologies that exist for TPV:

- TPV running in single server environment
- TPV running in a Network Deployment environment

In the first case, TPV runs in the JVM of the single server. This means that the collection and viewing of data all occurs in the single JVM. In the case of a Network Deployment setup, the collection of data occurs in the Node Agent and is stored in its memory. The data viewing occurs from the Deployment Manager server. This architecture helps the work to get distributed among the different nodes.

Some Performance Monitoring Infrastructure (PMI) data that can be monitored by TPV includes:

- JVM Heap utilization
- WebSphere pools and queues, such as database connection pool
- Application data, such as average servlet response time

**Configuring TPV settings**

The user and logging settings of TPV can be configured. These settings affect the performance of the application server. The activity monitoring of TPV can be configured on a per-user basis. Any changes made to TPV settings only affect the server being monitored and the user viewing the data.

**TPV user settings**

Select Monitoring and Tuning → Performance Viewer → Current Activity → server_name → Settings → User in the administrative console navigation tree.
The User settings will be displayed on the right side of the page. The settings are described in Table 5-1.

**Table 5-1  TPV User settings**

<table>
<thead>
<tr>
<th>TPV User settings</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refresh Rate</td>
<td>Specifies how frequently TPV collects data for server from the PMI service. The default is 30 seconds. The allowed range is 5 to 500 seconds.</td>
</tr>
<tr>
<td>Buffer Size</td>
<td>Specifies the number of entries to be stored for the server. Data displayed in TPV is stored in the short term memory buffer. After the buffer is full, each time a new entry is retrieved, the oldest entry is discarded. The default buffer size is 40 entries. Supported values are 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100. The larger the memory size, the more memory is consumed. Specify a buffer size that allows you to capture enough data without wasting memory.</td>
</tr>
</tbody>
</table>
| View Data As      | Specifies how values are displayed. Viewing options are:  
  - Raw Value: Displays absolute value. If the counter represents load data, such as the average number of connections in a database pool, then TPV displays the current value followed by the average, for example, 18 (avg:5).  
  - Change in Value: Displays the change in current value from the previous value.  
  - Rate of Change: Displays the ratio change/(T1 - T2), where change is the change in the current value from the previous value. T1 is the time when the current value was retrieved, and T2 is the time when the previous value was retrieved. |

The refresh rate and buffer size settings control how much temporal history you have for the application server. The default values provide you with a 20 minute history of the application server's performance. Changing one of these parameters affects the length of temporal history.
The values you set for refresh rate and buffer size depend on your use of TPV. To diagnose a known problem on a test machine, you may poll data more frequently and decrease your buffer size. For a production server, you may want to poll data less frequently and specify a buffer size depending on how much history you need. It is important to note that TPV is not intended to be a full-time monitoring solution.

**TPV log settings**

Select *Monitoring and Tuning → Performance Viewer → Current Activity → server_name → Settings → Log* in the console navigation tree. The log settings will be displayed on the right side of the page. The explanation of the settings can be found in Table 5-2.

<table>
<thead>
<tr>
<th>TPV log settings</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration</td>
<td>Specifies the length of time, in minutes, that logging continues, unless <strong>Stop Logging</strong> is clicked first. TPV is not intended as a full-time logging solution.</td>
</tr>
<tr>
<td>Maximum File Size</td>
<td>Specifies the maximum size, in megabytes, of a single file. Note that TPV automatically zips log files to save space and this parameter controls the pre-zipped file size and not the post-zipped, which is smaller.</td>
</tr>
<tr>
<td>Maximum Number of Historical Files</td>
<td>Specifies the number of files TPV writes before stopping. If TPV reaches the maximum file size before the logging duration ends, it continues logging in another file, up to the maximum. If TPV reaches the maximum number of historical files before the logging duration ends, TPV deletes the oldest historical file and continues logging in a new file. The total amount of data that is stored is constrained by the Maximum File Size and Maximum Number of Historical Files parameters.</td>
</tr>
<tr>
<td>File Name</td>
<td>Specifies the name of the log file. The server name and the time at which the log is started is appended to the log name to help users identify a log file.</td>
</tr>
<tr>
<td>Log Output Format</td>
<td>Specifies whether TPV writes log files as XML or in a binary format. We recommend binary format, as it provides a smaller log file when uncompressed.</td>
</tr>
</tbody>
</table>
TPV Advisor

The TPV Advisor provides advice and recommendations based on the collected PMI data to help tune the WebSphere system for optimal performance. In order to generate the advisor report, select Monitoring and Tuning → Performance Viewer → server_name → Advisor. This is shown in Figure 5-24.

The message specifies recommendations for performance tuning. Click the message to obtain more details. The data on the upper panel displays a summary of performance data for the server. The data corresponds to the same period that the recommendations were provided for. However, the recommendations may use a different set of data points during analysis than the set displayed in the summary page. Figure 5-25 on page 101 shows an example of what is obtained when clicking a message.

![Figure 5-24  TPV Advisor](image-url)
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 second table displays the average thread activity for the Web Container and Object Request Broker (ORB) thread pools. The activity is expressed as the number of threads or connections busy and idle.

Tuning WebSphere Application Server is a critical part of getting the best performance from your application. However, tuning WebSphere Application Server involves analyzing performance data and determining the optimal server configuration. This determination requires considerable knowledge about the various components in the application server and their performance characteristics. The performance advisors encapsulate this knowledge, analyze the performance data, and provide configuration recommendations to improve the application server performance. Therefore, the performance advisors provide a starting point to the application server tuning process and help you without requiring that you become an expert.
The advisors provide information about the following application server resources:

- Object Request Broker service thread pools
- Web container thread pools
- Connection pool size
- Persisted session size and time
- Data source statement cache size
- Session cache size
- Dynamic cache size
- Java virtual machine heap size
- DB2 Performance Configuration wizard

For example, consider the data source statement cache. It optimizes the processing of prepared statements and callable statements by caching those statements that are not used in an active connection (both statements are SQL statements that essentially run repeatable tasks without the costs of repeated compilation). If the cache is full, an old entry in the cache is discarded to make room for the new one. The best performance is generally obtained when the cache is large enough to hold all of the statements that are used in the application. The PMI counter, `prepared statement cache discards`, indicates the number of statements that are discarded from the cache. The performance advisors check this counter and provide recommendations to minimize the cache discards.

Using another example with pools in the application server, the idea behind pooling is to use an existing thread or connection from the pool instead of creating a new instance for each request. Because each thread or connection in the pool consumes memory and increases the context-switching cost, the pool size is an important configuration parameter. A pool that is too large can hurt performance as much as a pool that is too small. The performance advisors use PMI information about current pool usage, minimum or maximum pool size, and the application server CPU utilization to recommend efficient values for the pool sizes.

**TPV summary reports**

TPV provides five different summary reports that make important data quickly and easily accessible. View summary reports to help find performance bottlenecks in your applications and modules. The TPV summary reports are generated upon request and are not dynamically refreshed. The five reports you can display are:

- Servlets
- EJBs
- EJB Methods
- Connection Pool
In order to prepare a specific summary report, you must enable the minimum level of PMI data collection, or utilize the custom level of monitoring and enable specific counters.

To use the administrative console to view TPV summary reports:

1. Select Monitoring and Tuning → Performance Viewer → Current Activity → server_name → Summary Reports in the console navigation tree.
2. Click the application component or pool you would like to display.

**TPV performance modules**

TPV can be used to view Performance Monitoring Infrastructure (PMI) data in chart or table form. You view performance modules when your server is experiencing performance problems. For example, a common performance problem occurs when individual sessions are too large. To help view data on a session, you can view the Servlet Session Manager PMI module and monitor the SessionObjectSize counter to make sure that session object size is not too large.

To view the performance module data:

1. Select Monitoring and Tuning → Performance Viewer → Current Activity → server_name → Performance Modules in the console navigation tree.
2. Place a check mark in the check box beside the name of each performance module that you want to view. Expand the tree by clicking the plus sign (+) next to a node and shrink it by clicking the minus sign (−) next to a node.

   If you do not see all the PMI counters you expect, or a Performance Monitor you are interested in is visible but cannot be selected, PMI is not enabled for that particular counter or for any counter in that PM. Go to the PMI control area of the administrative console and enable PMI for any counters you wish to view, then return to TPV and select those PMs. To view the PMI page, select Monitoring and Tuning → Performance Monitoring Infrastructure → server_name.

3. Click View Modules. A chart or table providing the requested data is displayed on the right side of the page. Charts are displayed by default.

   Each module has several counters associated with it. These counters are displayed in a table underneath the data chart or table. Selected counters are displayed in the chart or table. You can add or remove counters from the chart or table by selecting or deselecting individual counters. By default, the first three counters for each module are shown. You can select up to 20 counters and display them in the TPV in the Current Activity mode.
4. Optionally, to remove a module from a chart or table, deselect the check box next to the module and click View Modules again.

5. Optionally, to view the data in a table, click View Table on the counter selection table. To toggle back to a chart, click View Graph.

6. Optionally, to view the legend for a chart, click Show Legend. To hide the legend, click Hide Legend.

Scaling the PMI data
You can manually adjust the scale for each counter so that the graph displays meaningful comparisons of different counters.

1. Find the counter whose scale you want to modify in the table beneath the chart.

2. Change the value for scale as needed:
   – When the scale is set to 1, the true value of the counter is displayed in the graph.
   – A value greater than 1 indicates that the value is amplified by the factor shown.
   – A value less than 1 indicates that the variable is decreased by the factor shown.

For example, a scale setting of .5 means that the counter is graphed as one-half its actual value. A scale setting of 2 means that the counter is graphed as twice its actual value. Scaling only applies to the graphed values and has no effect on the data displayed when viewing it in table form.

3. Click Update.

Clear values from tables and charts
1. Ensure that one or more modules are selected under Performance Modules in the TPV navigation tree.

2. Click Clear Buffer beneath the chart or table. The PMI data is removed from a table or chart.

Reset counters to zero
1. Ensure that one or more modules are selected under Performance Modules in the TPV navigation tree.

2. Click Reset to Zero beneath the chart or table. Reset to Zero sets a new baseline using the current counter readings at the instant the button is clicked. Future datapoints are plotted on the graph relative to their position at the time Reset to Zero is clicked. Datapoints gathered prior to the time Reset to Zero is clicked are not displayed, although they are still held in the TPV buffer. If Undo
Reset to Zero is clicked again, TPV displays all data recorded from the original baseline, not from the Reset to Zero point.

**Monitoring performance with TPV**

Once monitoring is enabled, TPV monitors the performance activity of all servers on a node, which can include the following:

- Application Servers
- Node agent for the node being monitored

TPV enables administrators and programmers to monitor the current health of WebSphere Application Server. Because the collection and viewing of data occurs in the application server, performance may be affected. To minimize performance impacts, monitor only those servers whose resources need to be optimized.

**Instructions**

1. Select **Monitoring and Tuning → Performance Viewer → Current Activity** in the administrative console. This is shown in Figure 5-26 on page 106.

2. Click the check box next to the server you would like to monitor and click **Start Monitoring**.

3. Click on the *server_name* link and a TPV console panel will be displayed, providing a navigation tree on the left and a view of real-time data on the current performance activity of a server on the right.

   From the navigation tree, you can select the server activity data that you would like to view. The options are:

   - **Advisor**
     
     Used to examine various data while your application is running. The Performance Advisor provides advice to help tune systems for optimal performance and gives recommendations on inefficient settings by using collected PMI data.

   - **Settings**
     
     User and logging settings for TPV can affect the performance of your application server.

   - **Summary Reports**
     
     View summary reports on servlets, enterprise beans (EJBs), EJB methods, connections pools, and thread pools in WebSphere Application Server.
Performance Modules

View performance modules that provide graphics and charts of various performance data on system resources, such as CPU utilization, on WebSphere pools and queues, such as database connection pools, and on customer application data, such as servlet response time. In addition to providing a viewer for performance data, TPV enables you to view data for other products or customer applications that have implemented custom PMI.

Figure 5-26 Start monitoring an application server using TPV built in the Admin Console
Performance module example

Rational Performance Tester (RPT) was used to generate application requests for the Trade V6.1 application with five users to our V6 server. The test ran for approximately three minutes. We produced a number of graphs by selecting different options in the performance module. Figure 5-27 shows the graph when JVM RunTime is checked. Figure 5-28 on page 108 shows it when a JDBC datasource is checked, and Figure 5-29 on page 109 shows it when particular JSPs are checked. This shows just some of the different graphs that can be generated by selecting different options in the performance module for TPV. Depending on what you wish to monitor, you can change these accordingly.

Figure 5-27 JVM heap utilization monitored by TPV
Figure 5-28  JDBC connection pool usage monitored by TPV
5.2.4 IBM Tivoli Composite Application Manager

*IBM Tivoli Composite Application Manager for WebSphere (ITCAM for WebSphere)* has evolved from several other products, namely *WebSphere Studio Application Monitor (WSAM)* and *IBM Tivoli OMEGAMON® XE for WebSphere*. ITCAM for WebSphere is part of the Tivoli's Composite Application Management solutions. ITCAM for WebSphere provides monitoring for WebSphere Application Server and J2EE applications, middleware adapters and transports, database calls, and back-end systems, such as CICS and IMS. It is a problem diagnostic and performance management tool for J2EE applications. The entire life cycle of a request can be examined and the amount of CPU time consumed by the requests.
ITCAM for WebSphere does not need to modify any J2EE or mainframe application code to collect data. The main sources for the data are from the Java Virtual Machine Tool Interface (JVMTI) and primitive, as well as the Performance Management Interface (PMI) in WebSphere, and the System Measurement Facility (SMF) in z/OS.

A large number of system level performance metrics are collected and can be viewed in real time using the ITCAM for WebSphere graphical console that is accessible through a Web browser. ITCAM for WebSphere reports on the status of the application servers and their CPU utilization, memory usage, and components, such as database connection pools, JVM thread pools, and EJB usage. ITCAM for WebSphere can bring attention to critical indicators by setting up thresholds within the product, which will send out e-mails as well as display critical information about the control panel.

Note: Much more information can be found in the following IBM Redbooks and guides:


Managing Server
ITCAM for WebSphere is a distributed application and consists of a Managing Server and one or more Data Collectors. The Managing Server controls and coordinates the Data Collectors and is the central point of access for monitoring. The Managing Server leverages WebSphere and requires either WebSphere Application Server 5.1x or 6.0.x. The Managing Server is available for the following platforms:

- AIX 5L™ V5.2
- AIX 5L V5.3
- Solaris™ 8 SPARC
- Solaris 9 SPARC
- Windows 2000 Server or Advanced Server with SP4
- Windows 2003 Server SE/EE
- Red Hat Enterprise Linux 3.0
- Red Hat Enterprise Linux 4.0
- SUSE Linux Enterprise Server 8
- SUSE Linux Enterprise Server 9
- Linux on IBM System z
The Managing Server also requires a database to be used as a repository for the data collected from the various systems. The databases that are supported are:

- DB2 8.1 FP6
- DB2 8.2
- Oracle® 8i SE R3 8.1.7
- Oracle 9i SE R2 9.2
- Oracle 10g

The Managing Server consists of several components:

- **Visualization Engine** is a J2EE application that runs in WebSphere and provides a graphical view of the monitoring environment.
- The **Kernel** registers components and Data Collectors that join the Managing Server and collects availability information.
- The **Publishing Servers** gather application and system event data from the Data Collectors, compute metrics, and implement the monitor and trap features.
- The **Archive Agents** receive the monitoring data from the Publishing Servers and store the data in the database.
- The **Global Publishing Servers** collects the data from the Publishing Servers and correlates multi-server requests.
- The **Message Dispatcher** is used to send messages using SNMP and e-mail protocols.
- The **Polling Agents** collect the data from Web servers for Apache 2.0 and above.

**Note:** The Visualization Engine requires Microsoft Internet Explorer®. Other Web browsers may have display or layout problems.
Figure 5-30 shows the relationship between the components in the Managing Server.

Figure 5-30   ITCAM for WebSphere Managing Server

Sizing of the Managing Server is dependent on a number of installation specific variables, such as the number of servers to monitor, transaction rate, and the monitoring level that is selected. Tests should be performed to size the Managing Server correctly at the level of monitoring that is needed.

For our tests, our Managing Server was installed on WebSphere V6.0.2.0 on AIX 5L V5.3 with a maximum heap of 512 MB. The Managing Server was running in an AIX logical partition (LPAR) with two CPUs and 8 GB of physical memory allocated to the partition in 64-bit mode. We were monitoring two WebSphere Application Servers on z/OS and had the monitoring level set to 1 for most of the tests.

Note: The Managing Server must be installed by an administrator on Windows or root on UNIX platforms. A user ID is required to manage the Managing Server and must be defined to the local operating system.
Data Collector for WebSphere Application Server

Data Collectors run inside of the application servers and use the native system services. The Data Collectors are tailored to the particular environments where they will be executed to take advantage of services that offered in a particular environment. The Data Collector for z/OS uses features like Cross-Memory Services and address fencing in MVS, which are not available on the distributed versions of the Data Collectors.

The Data Collectors run on the following platforms:

- AIX 5L V5.2
- AIX 5L V5.3
- Solaris 8 SPARC
- Solaris 9 SPARC
- HP-UX 11i 1
- Windows 2000 Server or Advanced Server with SP4
- Windows 2003 Server SE/EE
- Red Hat Enterprise Linux 3.0
- Red Hat Enterprise Linux 4.0
- SUSE Linux Enterprise Server 8
- SUSE Linux Enterprise Server 9
- Red Flag Advanced Server 4.0
- Red Flag Advanced Server 4.1
- OS/400® 5.2
- OS/400 5.3
- z/OS 1.4
- z/OS 1.5
- z/OS 1.6
- Linux on IBM System z

Note: The Data Collectors on Windows require the installation of Microsoft’s Services for UNIX (SFU). Installation on Windows 2003 requires that SFU be patched at level 8.01969.32 or higher.

The Data Collector is comprised of two components:

- The Command Agent collects request information about EJB invocations, database connection pools, thread pools, stack traces, memory analysis, and heap dumps.
- The Event Agent provides data to the Managing Server’s Publishing Server with information about system initialization data, application request level data, and application method level data.
The Data Collectors use *probes* in the application servers to analyze the performance of applications. The probes collect monitoring data and pass the data to the transports and then to the Managing Server. The Data Collectors do not analyze or interpret the data that is collected; they only pass the data to the Managing Server as quickly as possible.

ITCAM for WebSphere uses different resources to gather data:

- JVMTI garbage collection data, method trace, stack trace, CPU time, and heap dumps
- JMX™ system resources
- SMF system resources on z/OS
- PMI system resources
- OS services SCC, platform CPU, and its environment
- Byte Code Modification (BCM) instrumentation of some classes
Figure 5-31 shows a Data Collector configured in a WebSphere environment. The Tivoli Enterprise™ Management Agent is an optional software to manage many diverse systems using one console.

![ITCAM for WebSphere Data Collector architecture](image)

**Figure 5-31  ITCAM for WebSphere Data Collector architecture**

**Note:** ITCAM for WebSphere supports the IBM System z Application Assist Processor (zAAP). Data Collectors are also available for CICS and IMS.
**Administration**
Before monitoring can begin, it is necessary to do some basic setup functions within the Managing Server.

**Users and roles**
Anyone with authority can access the Visualization Server from a Web browser. Users can be granted access to the Visualization Server and can be given roles to restrict their ability to view certain statistics or reports. Adding a new user is done by selecting Administration → Account Management → User Profiles → Create User Account. Users do not have to be local operating system users, but must be associated with one. Users are assigned roles and groups that they are permitted to monitor.

Figure 5-32 shows the addition of a new user that has the role user for viewing reports under the WAS602x group.

![User Information Form](image)

**Figure 5-32  Adding or modifying a user**
Roles can be added by selecting **Administration → Account Management → Role Configuration → Create Role**. After entering the name of the new role, functionality is given by selecting the check box next to the function that is to be granted.

Figure 5-33 shows roles that currently exist in the system and the possibility to change what functions that the users that belong to the roles will be allowed to access.

![Table of Role Permissions](image)

**Figure 5-33  Setting role groups and their properties**

### Server management

The **Enterprise Overview** page within the Visualization Engine displays the high level health of a group of application servers. Grouping the servers allows for a quick view of the health of the enterprise. Reporting can be focused on a group of servers or an individual server. In a typical environment, application servers running the same applications would be grouped together.

A new server group can be added by selecting **Administration → Server Management → Server Groups → Create Group**. Characteristics about the server group are entered, such as what constitutes a slow response time, which servers belong to the group, and which users have access to the group.
Figure 5-34 shows a server group with characteristics that are specific to the group.

<table>
<thead>
<tr>
<th>GROUP INFORMATION</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>*Group Name</td>
<td>WAS602x</td>
</tr>
<tr>
<td>Description</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SERVER GROUP RESPONSE TIME - THRESHOLDS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter a percentage up to 10 times (999%) for each indicator's response.</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Indicator 1 (Slow Response)</td>
</tr>
<tr>
<td>Response Time</td>
<td>$\geq 25$ %</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PORTAL RESPONSE TIME - THRESHOLDS</th>
<th></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>BASELINE DEFINITIONS - Select one baseline definition and fill out the appropriate information.</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Rolling Date</td>
<td>7 days</td>
</tr>
<tr>
<td>Fixed Date (1-31 days)</td>
<td>Start Date: Jan 01 2002, End Date: Jan 01 2002</td>
</tr>
<tr>
<td>Fixed Response Time (0-10,000 ms)</td>
<td>Server Group</td>
</tr>
<tr>
<td>Portal</td>
<td>Gateway Servlet: 1000 (ms), Portal Pages: 1000 (ms), Authentication: 1000 (ms), Authorization: 1000 (ms), Model Building: 1000 (ms), Page Loading: 1000 (ms), Portlets: 1000 (ms)</td>
</tr>
</tbody>
</table>

*Figure 5-34 Server group parameters can be tailored for each group*
Unconfigured Data Collectors must be configured to start collecting data. This will make the Data Collector available to be placed within a server group. Configuring a Data Collector is done by selecting Adminstration → Server Management → Data Collector Configuration → Unconfigured Data Collectors. By selecting a configuration library, which will include or exclude certain classes from the data collected, checking the Data Collectors that is to be configured, and pressing the Apply button, the Data Collectors will become available for monitoring.

Figure 5-35 shows an unconfigured Data Collector being configured and given the J2EE Default configuration for classes to include and exclude during monitoring.

![Figure 5-35 Configuring a Data Collector](image)

Classes can be included or excluded within a Data Collector by configuring a Configuration Library. Configuration Libraries contain the Java package names and classes that are to be included or excluded during monitoring. Only one Configuration Library can be associated with a Data Collector. Wildcards are allowed in the package and classnames. The Configuration Libraries are configured from the Data Collector Configuration window by Configuration Library.

**Monitoring levels**

ITCAM for WebSphere uses three different monitoring levels to collect various amounts of data. The monitoring levels are:

- **L1 (Production Mode)** provides availability management, system resources, and basic request-level data. This monitoring level least affects the CPU overhead per transaction and is appropriate for servers that are not malfunctioning.
L2 (Problem Determination Mode) provides L1 monitoring and advanced request data, including external component and CPU information, as well as additional monitoring fields and functions. Under L2, component traces can be viewed. These traces provide J2EE request-related events that are made to external services. This level is used when there is a suspected problem or the need to capture data about external events but no need to capture all the method-level data.

L3 (Tracing Mode) is the most powerful monitoring level and utilizes all reporting elements that are available. More detail is displayed on the Server Activity window and the Request Detail page provides Method Trace with SQL statements. L3 has inherently higher overhead than the other monitoring levels and is used for servers that have been selected for diagnostics and detailed workload characterization.

**Note:** L2 or L3 must be selected to receive information about lock contentions and lock acquisitions.

**Note:** In the z/OS environment, a WebSphere server instance is represented by a Data Collector definition. It serves as a template for all the Data Collectors in the server regions belonging to the same server instance. The Data Collector configuration will be used by all Data Collectors in the server regions when monitoring the applications.

The monitoring level can be changed on-demand or by scheduling a time when the monitoring level will be changed. By selecting Administration → Monitoring On Demand®, the list of available servers and their current monitoring level is displayed. Selecting the arrow next to the server or server region allows for an immediate change of the monitoring level. A schedule can be appended to a server group by selecting a server region and assigning a previously created schedule.
Figure 5-36 shows the Monitoring On Demand window with the current levels of the server groups and servers.

A schedule allows for ongoing problems that occur during specific dates and times to have the monitoring level changed. A Schedule consists of one or more dates and times and monitoring levels. When assigned to a server region, the monitoring level will automatically be changed to the information specified in the schedule.
Figure 5-37 shows a new schedule being created that will start every day at 1600 and will change the monitoring level to L2. At 1830 every day, the monitoring level will be set to the system default.

Figure 5-37 Creating a new schedule for monitoring on demand

**Traps and alerts**

A *trap* is an occurrence of a condition or threshold. Traps can be set on application or server resources. When a trap is activated and the criteria in the trap is met, server actions can be taken, depending on the type of trap. For application traps, alert e-mails and SNMP messages can be sent, a thread dump can be collected, and a method trace can be collected. For server resource traps, alert actions include sending e-mails and sending SNMP messages. In both types of traps, the actions can be suppressed for a specified period of time as not to create too many e-mail or SNMP messages. Traps and alerts are configured by selecting **Problem Determination → Trap & Alert Management**.
Figure 5-38 shows a trap that will be triggered when the JVM CPU runs above 60% for more than five intervals. Once the trap is activated, it will send an SNMP message and will be considered an alert with “high” priority. Another trap will not be triggered for at least five minutes after the previous trap was triggered.

**Figure 5-38  Trap setup for high CPU utilization**
Monitoring with ITCAM

Monitoring, problem determination, and performance analysis are the main functions of ITCAM for WebSphere. Initially, the Enterprise Overview is presented with the Server Groups that have servers associated with them. The Enterprise Overview shows the transaction volume and response time for each server group. The transaction volume is the total of all transaction volumes for each server in the Server Group, and the response time shows the average response time across all servers in the group. The actual servers may have varying workloads. This view is beneficial when all servers in the cluster have an equal amount of work that is being distributed among them.

Figure 5-39 shows the enterprise view. In this example, there is one Server Group. The Server Group has two servers associated with it, and both are active. The view shows the number of transactions in the last hour among all servers, the throughput, and the average response time among all the servers.

![Enterprise Overview](image)

The Group Overview shows all the servers in a particular Server Group. A server may appear more than once if more than one servant is started under z/OS. This view shows the work being performed by each server and can be used to identify load balancing problems, or if a certain server is not performing as well as other servers. Clicking the server name drills down into the server details.
Figure 5-40 shows the servers associated with a specific Server Group. The display shows whether the server is active nor not, the total volume processed by each individual server, the server throughput, and each server’s average response time. This view gives a quick glance at load balancing and whether a server may not be performing as well as the other servers.

![Server Group Overview](image)

**Figure 5-40  Server Group Overview**

After selecting a server from the Server Group Overview, a more detailed view of the server selected is displayed. This gives more detail about the memory, CPU, utilization, number of sessions, thread pools, and database pools. The tool icons, located in each section, can be clicked to display other various information about the section.
Figure 5-41 shows the Server Overview display for the specific server selected. The information is more detailed than the Server Group Overview display, and can provide information about performance bottlenecks.

Figure 5-41   Server Overview

System Statistics Overview is a display that shows the details of servers in a Server Group or of a specific server from a Server Group. The view does not have graph representations of the data; rather, it displays the actual data. The difference between snapshots, which is defaulted to 15 seconds, can be displayed.

The view is configurable by clicking the Customize... button. This brings up other windows where different statistics that can be shown can be selected. This includes z/OS specific data, such as the processing time for zAAP.
Figure 5-42 shows the *Server Detail View* for a specific Server Group. There are only two servers specified; however, an extra servant for each server has been started to handle the increased workload. Looking at the statistics shows that the system is not ideally balanced, as the two servers in one node are using the maximum amount of CPU, while the others are using much less for processing.

![Server Detail View](image)

**Figure 5-42**  
**Server Detail View**

Selecting a toolbox from one of the servers will display the *System Activity Display*. This display will show any currently processing requests, which can be looked at in detail, recent request with response and accumulated CPU time, and lock contentions. The active requests can be browsed to the method level, if necessary, and provides in-depth information concerning the life cycle of a request.

Figure 5-43 shows an active request. The request can be expanded to view the flow of the request, including specifics about the processing times for components of a request.

![Active requests](image)

**Figure 5-43**  
**Active requests**
The *System Resource Overview* is a display that shows the details for each component of every request. The display is based on the server and is viewed by either selecting the toolbox graphic from previous pages and selecting **System Resources** or by selecting **Availability → System Resources**. The display shows graphics and information based on the components being used. The display can be switched between PMI data and SMF data. The following components can be examined:

- EJBs
- JCA Connection Pools
- JVM/System
- DB Connection Pools
- JTA Transactions
- ORB
- Session Manager
- Thread Pools
- Web Applications
- SQL
- JCA-CICS
- MQI
Figure 5-44 shows the System Resource Overview of a single server. The PMI data is shown, and mousing over some of the graphics produces more detailed information about the component. Clicking the headings of the graphics will provide more detail about the component.

Note: Some of the components require that the monitoring level be above L1 to have data captured and displayed.

ITCAM for WebSphere includes several components that can help in determining a problem with memory. There is memory analysis, heap analysis, and memory leak analysis available to view. These components are available by selecting Problem Determination → Memory Diagnosis → item of interest.
The memory analysis component analyzes either garbage collection or Java heap size against numerous other measurements, such as response time, over a period of time, and presents a graph with the data. This information is used, for example, to determine if garbage collection is causing slow response time or if garbage collection is happening too often, which may be indicative of other problems.

Figure 5-45 shows total garbage collection times against the average response time over a one hour period. Even though there was a spike of over six seconds for garbage collection, the average response time was below 150 ms.

<table>
<thead>
<tr>
<th>Minutes Ago</th>
<th>Total GC Time</th>
<th>Avg. Response Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>393 ms</td>
<td>0 ms</td>
</tr>
<tr>
<td>2</td>
<td>300 ms</td>
<td>0 ms</td>
</tr>
<tr>
<td>3</td>
<td>360 ms</td>
<td>0 ms</td>
</tr>
<tr>
<td>4</td>
<td>404 ms</td>
<td>0 ms</td>
</tr>
</tbody>
</table>

Figure 5-45 Comparing garbage collection times to response times
Heap analysis displays the current objects in a heap, number of instances, total size, percentages of total size, and total instances. This information can be used in several ways. The information can be used to determine if a memory leak is occurring, if objects are not being collected, or a possible application error is causing objects to grow.

Figure 5-46 shows a Java heap. The heap shows that there are no really large objects and that character arrays are taking up the most space.

<table>
<thead>
<tr>
<th>Class name</th>
<th>Total size (kb)</th>
<th>Percent of total size</th>
<th># of instances</th>
<th>Percent of total #</th>
</tr>
</thead>
<tbody>
<tr>
<td>char[]</td>
<td>15166</td>
<td>38.0 %</td>
<td>123714</td>
<td>15.0 %</td>
</tr>
<tr>
<td>byte[]</td>
<td>4901</td>
<td>11.0 %</td>
<td>24288</td>
<td>3.0 %</td>
</tr>
<tr>
<td>int[]</td>
<td>698</td>
<td>1.0 %</td>
<td>9384</td>
<td>1.0 %</td>
</tr>
<tr>
<td>org/clip/itm/core/impl/AttributeImpl</td>
<td>237</td>
<td>0.0 %</td>
<td>2098</td>
<td>0.0 %</td>
</tr>
<tr>
<td>long[]</td>
<td>232</td>
<td>0.0 %</td>
<td>745</td>
<td>0.0 %</td>
</tr>
<tr>
<td>org/clip/itm/core/impl/BasicObjectImpl$EPropertiesHolderImpl</td>
<td>199</td>
<td>0.0 %</td>
<td>7301</td>
<td>0.0 %</td>
</tr>
<tr>
<td>UNKNOWN_OBJECT[]</td>
<td>157</td>
<td>0.0 %</td>
<td>4</td>
<td>0.0 %</td>
</tr>
<tr>
<td>org/clip/itm/core/impl/EReferenceImpl</td>
<td>142</td>
<td>0.0 %</td>
<td>1234</td>
<td>0.0 %</td>
</tr>
</tbody>
</table>

Figure 5-46  Java heap showing object characteristics

Memory leaks can bring systems down quickly. ITCAM for WebSphere includes a component that can show the difference between two different Java heap dump snapshots. The snapshots are compared and their changes are displayed. By looking at the data being displayed, it is easier to determine if a memory leak is occurring and which objects are causing the memory leak.
Figure 5-47 shows a memory leak analysis of two heap dumps and their differences. In this example, there is no potential memory leak, as there has been minimal change between the Java heap dumps.

**Heap Properties**

<table>
<thead>
<tr>
<th>App Server</th>
<th>FECELLA07/PENODEB peel02b (L3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap 1 Snapshot</td>
<td>May 31, 2006 11:15:43 AM</td>
</tr>
<tr>
<td>Size of Live Objects on Heap(MB)</td>
<td>40 (42999996 bytes)</td>
</tr>
<tr>
<td># of Objects in Heap</td>
<td>765394</td>
</tr>
<tr>
<td>GC</td>
<td>Yes</td>
</tr>
</tbody>
</table>

| Heap 2 Snapshot | May 31, 2006 11:18:00 AM       |
| Size of Live Objects on Heap(MB) | 42 (44454596 bytes)              |
| # of Objects in Heap | 613226                           |
| GC | Yes |

**Heap Comparison Results Table**

```
+-------------------------------+-------------------+-------------------+-------------------+-------------------+
<table>
<thead>
<tr>
<th>Class name</th>
<th>Original # of instances</th>
<th>Original Total Size (kb)</th>
<th># of instances</th>
<th>Total size(kb)</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte[]</td>
<td>15837</td>
<td>5058</td>
<td>666</td>
<td>591</td>
</tr>
<tr>
<td>char[]</td>
<td>128268</td>
<td>15281</td>
<td>5136</td>
<td>372</td>
</tr>
<tr>
<td>int[]</td>
<td>8164</td>
<td>622</td>
<td>2203</td>
<td>19</td>
</tr>
<tr>
<td>boolean[]</td>
<td>3889</td>
<td>30</td>
<td>1286</td>
<td>9</td>
</tr>
<tr>
<td>long[]</td>
<td>618</td>
<td>231</td>
<td>158</td>
<td>5</td>
</tr>
<tr>
<td>short[]</td>
<td>86</td>
<td>12</td>
<td>405</td>
<td>5</td>
</tr>
<tr>
<td>int[]</td>
<td>205</td>
<td>6</td>
<td>91</td>
<td>1</td>
</tr>
<tr>
<td>org.eclipse.emf.ecore.SimpleResourceImpl</td>
<td>2098</td>
<td>237</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>org.eclipse.emf.ecore.EObjectImpl</td>
<td>7373</td>
<td>201</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>UNKNOWN_OBJECT[]</td>
<td>4</td>
<td>157</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>org.eclipse.emf.ecore.ESimpleEObject</td>
<td>1234</td>
<td>142</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>org.eclipse.emf.ecore.ESimpleEObject</td>
<td>995</td>
<td>111</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
+-------------------------------+-------------------+-------------------+-------------------+-------------------+
```

**Figure 5-47 Memory leak analysis of two heap dumps**

ITCAM for WebSphere includes a component for viewing a snapshot of the threads running in the Java Virtual Machine (JVM). When a system becomes unresponsive, a thread dump can help determine for which resource threads are waiting. The entire thread dump can be viewed in Windows text editor by selecting the **Thread Dump** button.
Figure 5-48 shows the threads displayed from a particular WebSphere server. The thread details can be viewed by selecting the thread. The details include the stack trace for the thread.

<table>
<thead>
<tr>
<th>SERVER PROPERTIES</th>
<th>THREAD DUMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Snapshot Date</td>
<td>May 31, 2006</td>
</tr>
<tr>
<td>Snapshot Time</td>
<td>11:40:21 AM</td>
</tr>
<tr>
<td>Application Server Name</td>
<td>pesr01a</td>
</tr>
<tr>
<td>Application Server IP Address</td>
<td>0.12.4.38</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ACTIVE THREADS</th>
</tr>
</thead>
<tbody>
<tr>
<td>/system/</td>
</tr>
<tr>
<td>Name</td>
</tr>
<tr>
<td>main</td>
</tr>
<tr>
<td>RMI Runtime</td>
</tr>
<tr>
<td>GC event write thread</td>
</tr>
<tr>
<td>Finalizer</td>
</tr>
<tr>
<td>GC Daemon</td>
</tr>
<tr>
<td>GC Helper 1</td>
</tr>
<tr>
<td>GC Helper 2</td>
</tr>
<tr>
<td>GC Helper 3</td>
</tr>
<tr>
<td>Reference Handler</td>
</tr>
<tr>
<td>RMI ConnectionExpiration (wsap1.https.ibm.com:9119.com/cyana.kernel.rmi.CmClientSocketFactory@dd5)</td>
</tr>
<tr>
<td>RMI ConnectionExpiration (wsap1.https.ibm.com:9119.com/cyana.kernel.rmi.CmClientSocketFactory@dd5)</td>
</tr>
<tr>
<td>RMI ConnectionExpiration (wsap1.https.ibm.com:9119.com/cyana.kernel.rmi.CmClientSocketFactory@dd5)</td>
</tr>
<tr>
<td>RMI ConnectionExpiration (wsap1.https.ibm.com:9119.com/cyana.kernel.rmi.CmClientSocketFactory@dd5)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>THREAD GROUP PROPERTIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>Active Thread Count</td>
</tr>
<tr>
<td>Active Thread Group Count</td>
</tr>
<tr>
<td>Max Priority</td>
</tr>
<tr>
<td>Daemon Thread Group</td>
</tr>
<tr>
<td>Destroyed</td>
</tr>
</tbody>
</table>

Figure 5-48  Displaying the threads of a JVM

Performance analysis

The previous section was based on a snapshot of the data from the Data Collectors. There is data that is being archived and reports can be generated from this data. The data that is stored is transaction data that has been passed from the Publishing Agent to the Archive Agent and is subject to filtering based on the sampling rate specified for the monitoring level.

Reports are broken down into Application Reports, Server Reports, and Daily Statistics. Reports can be generated and based on any Server Group or all Server Groups and any Server or all Servers.

The Application Reports that can be generated are:

- Request/Transaction provides a clear overview of the behavior of the application server. The report can generate metrics based on throughput, response time, and CPU time, and can be further restricted based on request/transaction type and name.
- **Method/Program** provides the performance of the methods in the requests that have been processed by the application server. The report can generate metrics based on throughput, response time, and CPU time, and can be further restricted based on request/transaction type and name.

- **SQL** provides information concerning the performance of SQL calls in the requests that have been processed by the application server. The report can generate metrics based on throughput and response time, and can be further restricted based on request/transaction type and name, and SQL call and database table name.

- **MQI** provides information concerning the performance of MQ calls in the requests that have been processed by the application server. The report can generate metrics based on throughput, and response time, and can be further restricted based on request/transaction type and name, and MQI call, Queue Manager, and Queue name.

- **Lock Analysis** provides the lock history of in-flight transactions. The report can generate metrics based on number of lock acquisitions, number of lock contentions, and total acquisition time, and can be further restricted based on request/transaction type and name.

- **Portal** provides details about the performance of WebSphere Portal Servers.

- **Top Reports** are a quick and convenient way to run a report for request, method, or SQL data. Top Reports provide the top 100 result records for the selected metric.

Most of the reports have areas that can be drilled down and produce other types of reports based on the data selected. The level of detail is based on the reporting level. It is possible to drill down a request from JSPs used, servlets, EJBs, and database calls. The reports can be saved for future reference. The reports may be viewed as a Portable Document Format (PDF), e-mailed to another user, or saved for future reference.
Figure 5-49 shows a trend report that is reporting on the response times based on the hour of the day. The report breaks the information down into a graph and data table. The average response time is shown for the period selected. Clicking the hour of the day will create a decomposition report.

**Figure 5-49  Trend report of response times versus hour of the day**
Figure 5-50 shows a decomposition report of the requests that were performed during the hour selected from the trend report. The graph and data table represent the same data. The data can further be examined by selecting the requests.

The Server Reports consist of the following reports:

- The System Resources Report provides reports based on memory utilization and database connectivity for the application servers.
- The Server Availability Report provides a report on the percentage of server availability. In a Server Group, availability is defined as the total amount of time when one or more servers of the group are running divided by the total elapsed time.
The *Capacity Analysis Report* provides information that is used to evaluate the capacity of the system using supply and demand metrics.

These reports are used for determining if the hardware being used is the amount necessary to cover the response and availability that is required.

Figure 5-51 shows a System Resource Trend Report of heap utilization during the hours of the day. This information is useful for discovering peaks in operation and addressing them as needed.
Figure 5-52 shows the usage of a heap by a server during a specific time period that was selected on the previous window. The report shows that the memory utilization is not the same between the two servers, which could indicate that one server is doing more work.

Any of the reports can be saved or scheduled and viewed at a later time. This information is a good source for understanding if there are bottlenecks that are arising, or if business is growing faster than the hardware.
5.3 Other techniques to analyze performance problems

Another technique to analyze performance problems, besides using the tools discussed in 5.2, “Monitoring and profiling tools” on page 58, is to use dynamic MVS console commands. We will discuss this further in 5.3.1, “Dynamic MVS console commands” on page 139.

5.3.1 Dynamic MVS console commands

There are various dynamic MVS console commands that can be used as a guide to determine how your application server is doing. They give you limited information, but can be a quick way to check if anything is going wrong, and if so, other tools described in this book can be used to determine exactly what is going on.

In order to see if your application server is processing requests, or only certain type of requests, or if it is hung, you can issue the following command:

F <ServerName>,DISPLAY,WORK,ALL

Example 5-8 shows the output of this command.

Example 5-8  Sample output of DISPLAY,WORK,ALL command

F PFSR03B,DISPLAY,WORK,ALL
BB000255I TIME OF LAST WORK DISPLAY Mon Jun  5 14:17:43 2006
BB000256I TOTAL EJB REQUESTS 0  (DELTA 0)
BB000257I CURRENT EJB REQUESTS 0
BB000258I EJB REQUESTS IN DISPATCH 0
BB000267I TOTAL EJB TIMEOUTS 0  (DELTA 0)
BB000256I TOTAL SERVLET REQUESTS 2143 (DELTA 1105)
BB000257I CURRENT SERVLET REQUESTS 74
BB000258I SERVLET REQUESTS IN DISPATCH 40
BB000267I TOTAL SERVLET TIMEOUTS 0  (DELTA 0)
BB000256I TOTAL MDB REQUESTS 0  (DELTA 0)
BB000257I CURRENT MDB REQUESTS 0
BB000258I MDB REQUESTS IN DISPATCH 0
BB000267I TOTAL MDB TIMEOUTS 0  (DELTA 0)
BB000188I END OF OUTPUT FOR COMMAND DISPLAY,WORK,ALL
The response to the DISPLAY,WORK,ALL command consists of three sections:

- EJB totals
- Servlet totals
- MDB totals

Example 5-8 on page 139 shows that only servlet work is currently processing. Note that if you drive an HTTP request that runs a JSP or servlet that calls an EJB, only the servlet count increments. The counts are, by protocol, used to get to the controller.

Each section consists of four counts. Let us look at these based on servlet output:

- **BBOO0257I TOTAL SERVLET REQUESTS 2143 (DELTA 1105)**
  Total number of requests processed since the server was initialized (in this case, 2143). DELTA is the number of requests processed since the last time this display-work command was issued (in this case, 1105 requests were processed since 14:17). Therefore, in order to see if your server is hanging and not processing any work, or it is processing the requests very slowly, issue the command multiple times and pay attention to the DELTA count.

- **BBOO0258I SERVLET REQUESTS IN DISPATCH 40**
  Work currently being processed by the servant(s). If your servants are processing at their highest capacity, you should see this number equal to number of active servants * number of worker threads per servant. In the example above, there was one active servant with 40 worker threads defined. Since the count is 40, that means that all worker threads were processing and none was idle waiting for work.

- **BBOO0257I CURRENT SERVLET REQUESTS 74**
  The total number of requests that came to the server and have not yet been completed. This count is the sum of IN DISPATCH (see previous bullet) and the work that is waiting on WLM queue, as well as work that is waiting to go back to the client.
  - If the IN DISPATCH count equals the CURRENT count, that means that all requests are currently processing and there are no delays outside the servant(s).
  - If the IN DISPATCH count is not at its capacity and the CURRENT number is higher than IN DISPATCH, then the requests are either not being dispatched by WLM quickly enough (a possible change to WLM settings is needed) or a controller is not able to manage the requests/responses back to the client (further review of what the controller is doing is needed).
– If the IN DISPATCH count is at its capacity and the CURRENT count is higher than IN DISPATCH, then it is very likely that the requests are backing up on WLM queue either because the server cannot handle the workload that is coming in or the requests’ dispatch in the servant(s) is taking a long time to process.

▶ BBOO0267I TOTAL SERVLET TIMEOUTS 0 (DELTA 0)

The number of requests that timed out for this application server since the server was initialized. DELTA is the number of timed out requests since the last time this display-work command was issued.

The interpretation of the counts in the output can be influenced by different factors. The explanation above gives you an idea what these number could mean. One factor that will greatly influence how you interpret the counts is whether you use work that requires affinity (see 9.3.2, “Temporal affinity workload balancing” on page 227). Another command can give you a more granular look at individual servants. This will show you if all servants are processing work:

F <ServerName>,DISPLAY,WORK,SERVLET,SRS

Example 5-9 shows an the output of this command.

Example 5-9 Sample output of ‘display,work,servlet,srs’ command

<table>
<thead>
<tr>
<th>Command</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>F PFSR03B,DISPLAY,WORK,SERVLET,SRS</td>
<td></td>
</tr>
<tr>
<td>BB0002551 TIME OF LAST WORK DISPLAY Fri Jun 2 14:15:58 2006</td>
<td></td>
</tr>
<tr>
<td>BB0002591 STC26663: TOTAL SERVLET REQUESTS 0 (DELTA 0)</td>
<td></td>
</tr>
<tr>
<td>BB0002601 STC26663: SERVLET REQUESTS IN DISPATCH 0</td>
<td></td>
</tr>
<tr>
<td>BB0002591 STC26650: TOTAL SERVLET REQUESTS 176 (DELTA 44)</td>
<td></td>
</tr>
<tr>
<td>BB0002601 STC26650: SERVLET REQUESTS IN DISPATCH 40</td>
<td></td>
</tr>
<tr>
<td>BB0002591 STC26652: TOTAL SERVLET REQUESTS 0 (DELTA 0)</td>
<td></td>
</tr>
<tr>
<td>BB0002601 STC26652: SERVLET REQUESTS IN DISPATCH 0</td>
<td></td>
</tr>
<tr>
<td>BB0001881 END OF OUTPUT FOR COMMAND DISPLAY,WORK,SERVLET,SRS</td>
<td></td>
</tr>
</tbody>
</table>

The servants are identified by STC number (Started Task). You can match this number to a specific ASID by:

▶ Looking at the joblog output
▶ Looking at the SDSF.DA (Active Users) output - JobID column
▶ Issuing the following MVS console command:

f <ServerName>,display,servants

In Example 5-9, there were three servants running (three different STC numbers in the output) and only one was handling servlets requests.
If you determine that there might be a delay either in the controller and in the servant (the requests are not processing as quickly as expected, for example), you can issue another command that will generate Java tracebacks for all threads in the controller and servant. The output will not go to the console, but to the SYSPRINT part of the joblog output for each individual task:

F <ServerName>,STACKTRACE

The output of the STACKTRACE command in the SYSPRINT will begin with this entry:

==================================
--- Java Thread Stack Tracebacks ---
==================================

In the servants, you can identify WebSphere worker threads by the following line at the bottom of the traceback:

com.ibm.ws.util.ThreadPool$ZOSWorker.run(ThreadPool.java:1688)

A worker thread that is idle (available for processing the next request) will have the following traceback:

Thread.WebSphere:ORB.thread.pool t=008cc470,5,main.
com.ibm.ws390.orb.CommonBridge.nativeRunApplicationThread(NativeMethod)
com.ibm.ws390.orb.CommonBridge.runApplicationThread(Unknown Source)
com.ibm.ws.util.ThreadPool$ZOSWorker.run(ThreadPool.java:1688)

Note that in some situations, the traceback will not format completely. These are also Java tracebacks only, so they will not contain non-Java information. If this information is required, an MVS console dump should be collected.

There are different variations of the display,work command. To learn about all the options, add word help at the end of the command to see all the option (see Example 5-10). You can also review the InfoCenter for other useful dynamic commands.

Example 5-10 Sample help commands

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>f &lt;ServerName&gt;,display,work,help</td>
</tr>
<tr>
<td>f &lt;ServerName&gt;,display,help</td>
</tr>
<tr>
<td>f &lt;ServerName&gt;,help</td>
</tr>
</tbody>
</table>
After having discussed our infrastructure, our sample workload, and using workload drivers and monitoring tools in Part 1, “Our environment” on page 7, we will now get into the extensive area of best practices for setting up the infrastructure for performance. We expect that the information provided in this part of the book can serve as a good checklist for both preventive and corrective activities in the area of performance.

The part is organized as follows:

- Chapter 6, “How to determine the area of interest” on page 145 is a high-level introduction to the typical tasks to be performed in case of performance problems. We are addressing each of those tasks further in this part of the book.
In Chapter 7, “WebSphere Application Server and HTTP Server” on page 159, we address all important parameters and other artifacts in the IBM HTTP Server for z/OS and WebSphere Application Server for z/OS that influence performance to some extent.

The performance of a WebSphere Application Server servant region is very dependent on the way the JVM behaves in terms of memory usage. Chapter 8, “Java Virtual Machine” on page 191 focuses on the workings of the Java Virtual Machine (JVM) and the Garbage Collection (GC) in SDK 5.0.

Chapter 9, “z/OS subsystems” on page 217 addresses all the important settings, parameters, and other information that could help in optimizing the z/OS subsystems being used by WebSphere Application Server.

In many cases, WebSphere Application Server applications access back-end databases and transaction systems. Also, applications may use WebSphere MQ to get to those back-end systems. In Chapter 10, “Connectors” on page 237, we talk about best practices in using Java Database Connectivity (JDBC) for DB2 access, J2C connectors for CICS and IMS access, and JMS for accessing WebSphere MQ.

**Note:** In this part of the book, we talk about connectivity from an infrastructure standpoint. In Chapter 12, “Accessing EIS resources” on page 301, we will talk about connectivity from an application development perspective.
How to determine the area of interest

This chapter covers several common problems that could occur when running an application in WebSphere Application Server and how to determine where to look. The topics covered are:

- System configuration before deployment, discussed in 6.2, “Configuration check” on page 146
- Typical problem areas, discussed in 6.3, “Problem areas” on page 148
- Slow response from the Application Server, discussed in 6.4, “Slow response from the application server” on page 151
- No response from the Application Server, discussed in 6.5, “No response from the application server” on page 152
- High CPU utilization, discussed in 6.6, “High CPU utilization” on page 154
- High memory utilization, discussed in 6.7, “High memory utilization” on page 155
6.1 Introduction

Performance problems can arise in many different shapes and forms. The most common performance problems are that the application server has stopped responding, the response time for certain requests is not acceptable, the memory utilization is poor, or the application server is using a (too) large amount of CPU time. This section will try to answer the question of “where do I look?” when a performance problem occurs. One performance problem may be masking another problem. It is important to solve all performance problems and not just one that makes performance unacceptable.

The performance problem may not be application server related, but related to hardware, the operating system, external sources, or the application itself that is deployed to the application server. It is important to note that the response time of an application cannot be faster than its slowest component.

6.2 Configuration check

It is important to have a methodology about how to determine where to look and what should be changed before starting to tweak different systems. There are three distinct tuning areas for WebSphere on z/OS:

- z/OS itself
- WebSphere Application Server
- The application

Before deploying an application in production, it is important to have a system that is tuned well. One way to make sure that you will get a well-tuned infrastructure could be to deploy a sample workload, such as Trade V6.1, and execute a variety of load tests with this sample workload, while using monitoring tools and techniques at the same time to narrow down eventual bottlenecks.

The next section shows a list of z/OS components that require attention before deploying an application to WebSphere Application Server.

6.2.1 z/OS components to check

- Virtual and real storage
- System Monitoring Facility (SMF)
- Resource Recovery Service (RRS)
- Workload Manager (WLM)
6.2.2 WebSphere Application Server components to check

This is a list of WebSphere components that require attention before deploying an application in WebSphere:

- Java Virtual Machine (JVM)
- Object Request Broker (ORB)
- Dynamic caching
- EJB container
- Session management
- Data sources
- Tracing
- Security
- SOAP and XML parsing
- Web container
- Connectors
- Plug-in configuration
- HTTP server

**Note:** Information about WebSphere Application Server tuning can be found in Chapter 7, “WebSphere Application Server and HTTP Server” on page 159 and Chapter 10, “Connectors” on page 237.
6.2.3 Application components to check

This is a list of application components that require attention before deploying the application in WebSphere:

- Local and remote references
- Synchronized methods
- Access intent
- Object size
- Object caching
- Data structures
- XML parsing
- Aggregated business objects
- Code optimization.

**Note:** Information about application tuning can be found in Part 3, “Best practices - application” on page 267.

6.3 Problem areas
There are numerous areas in an infrastructure that can cause problems with performance. The problems may be related to hardware, software, or operating system settings. In many cases, the application is the cause of the performance problems. Figure 6-1 shows a typical WebSphere environment. Any of the pieces shown in the diagram could cause a failure or slowdown in performance. Connected systems, such as CICS or DB2, may also be a cause of a performance problem.

In most cases, the best approach is to work backwards. In a simple example where an application is connected by HTTP and uses JSPs, servlets, EJBs, and database connectivity, it would make sense to start looking at the database first. If the database is tuned properly, then move backwards to the JDBC connection and examine the properties of the data source and the performance of the various aspects of the data source, such as number of connections used and concurrent waiters. Afterwards, the EJBs and EJB container can be investigated for performance issues, such as access intent, transactions, and thread pools. Moving backwards to the servlets, JSPs, and Web container items, such as servlet caching, how EJBs are called (local or remote), number of JSP parts, and the number of container threads can lead to clues why performance is slow.

**Figure 6-1   A WebSphere environment**
The performance problem could lie outside of WebSphere Application Server and z/OS completely. There could be a problem with the IP sprayers or the firewall could be set up incorrectly. The HTTP server could be a bottleneck. All of these must be considered when looking into performance problems.

If none of these items yield clues to why performance is slow under WebSphere, then the WebSphere Application Server work can be displayed from the z/OS console. Information can be displayed about:

- Active controllers
- Servants
- Sessions
- Trace settings
- Java trace
- JVM head statistics
- Work elements
- Error log

Thread dumps, heap dumps, or an SVC dump on z/OS may be necessary to determine certain types of problems, such as memory leaks or a hung system. A thread dump is a list of all active thread and monitors running in the JVM. Thread dumps are very useful for determining problems with a hung application server. A heap dump is a snapshot of all objects in a JVM, their size, and reference. Heap dumps help identify memory leak problems. An SVC dump provides a snapshot of the virtual storage under z/OS and contain a summary dump, control blocks, and other system code.

**Note:** Several tools are available for z/OS to help understand heap, thread, and SVC dumps. These are described in the following IBM Redpaper:


The Help System supplied with WebSphere for z/OS contains a wealth of information about tuning, performance, and troubleshooting. This information should be used in conjunction with the information in this chapter.
6.4 Slow response from the application server

A slow responding application is one of the most common problems; it is also a problem that could have a myriad of causes. Some of the causes are due to overloaded systems or spikes in activity that were not expected. Sizing of the hardware and software is important, and it is very important to consider growth and spikes in activity. This problem can be duplicated by running load tests on the system and determining when the response is no longer acceptable.

Other problems involve one or more components that are not configured for optimal performance. Within WebSphere Application Server, certain operations are much heavier in CPU utilization and take more time, for example, XML parsing, which is also used in Web services. In many cases, the application is the cause of the performance problem. The application must be able to scale and leverage the underlying WebSphere technologies efficiently.

Performance problems vary from one another. These are steps that can be taken to investigate why performance is slow:

- The software should be up to date. IBM releases fixes for WebSphere and the problem that is being experienced may have been corrected. Fixes, documentation, and other information can be found on the WebSphere Application Server for z/OS support page at:
  

- Examine the WebSphere Application Server and z/OS log files. The logs may provide information about the problem being experienced. More information can be found in Chapter 7, “WebSphere Application Server and HTTP Server” on page 159 and Chapter 9, “z/OS subsystems” on page 217.

- Work backwards from the last system used to the first one used. This includes all subsystems.

- Determine if the thread pools being used are adequate. The highest number of threads should be in the first component that is used and the lowest number should be the last component used. This makes a funnel effect and keeps threads from doing work and then waiting for another component. In the case where there are multiple entry points into the application, then a component in the middle may have more threads. More information can be found in Chapter 7, “WebSphere Application Server and HTTP Server” on page 159.
Make sure there is enough of a Java heap to run the application and that the expansion and contraction of the heap are not causing the slowdowns. Use a single heap size, if possible. Changing the heap size will require a restart of the application server. More information can be found in Chapter 8, “Java Virtual Machine” on page 191.

Examine the garbage collector. The JVM is frozen every time a full garbage collection is done. Enabling verbose garbage collection requires a restart of the application server. More information can be found in Chapter 8, “Java Virtual Machine” on page 191.

Examine the real storage on z/OS. Memory paging slows down the system. More information can be found in Chapter 9, “z/OS subsystems” on page 217.

Use the MODIFY commands in z/OS for WebSphere and examine their output. More information about which MODIFY commands are relevant is described in Chapter 7, “WebSphere Application Server and HTTP Server” on page 159.

Use thread and z/OS SVC dumps to examine which component is being held. Tools are necessary to analyze the output of these dumps. More information can be found in Chapter 7, “WebSphere Application Server and HTTP Server” on page 159 and Chapter 9, “z/OS subsystems” on page 217.

Workload activity reports provides a transaction response time analysis by MVS Workload Manager (WLM) service and report class. The reports are described in detail in Chapter 9, “z/OS subsystems” on page 217.

Test the components separately, if possible, to locate which components have slow response times. This may require writing separate drivers or specific load testing to test the components.

Examine the application for performance bottlenecks. Many performance bottlenecks are caused by problems in the application. More information can be found in Part 3, “Best practices - application” on page 267.

### 6.5 No response from the application server

The next most common problem is that WebSphere Application Server has stopped responding to requests. In most cases, all the threads in the Web container or EJB container are waiting on a resource. This could be due to a resource that is not responding or is slow in responding, and all the threads in a container are waiting for that particular resource, such as a database connection. In some cases, the application logic causes the thread never to release.
Performance problems vary from one another, these are steps that can be taken to investigate why the application server has stopped responding:

- The software should be up to date. IBM releases fixes for WebSphere and the problem that is being experienced may have been corrected. Fixes, documentation, and other information can be found on the WebSphere Application Server for z/OS support page at:
  

- Examine the WebSphere and z/OS log files. The logs may provide information about the problem being experienced. More information can be found in Chapter 7, “WebSphere Application Server and HTTP Server” on page 159 and Chapter 9, “z/OS subsystems” on page 217.

- Take a thread dump to determine for which resource the threads are waiting and on which resource they are waiting. Several tools can be used to examine the thread dumps. More information can be found in Chapter 7, “WebSphere Application Server and HTTP Server” on page 159.

- Take a SVC dump to get more detailed information about the problem. More information can be found in Chapter 9, “z/OS subsystems” on page 217.

- Use the MODIFY commands in z/OS for WebSphere and examine their output. More information about which MODIFY commands are relevant are described in Chapter 7, “WebSphere Application Server and HTTP Server” on page 159.

- Workload activity reports provides a transaction response time analysis by MVS Workload Manager (WLM) service and report class. The reports are described in detail in Chapter 9, “z/OS subsystems” on page 217.

- Determine if the thread pools being used are adequate. The highest number of threads should be in the first component that is used and the lowest number should be the last component used. This makes a funnel effect and keeps threads from doing work and then waiting for another component. In the case where there are multiple entry points into the application, then a component in the middle may have more threads. More information can be found in Chapter 7, “WebSphere Application Server and HTTP Server” on page 159.

- Make sure there is a enough Java heap to run the application and that the expansion and contraction of the heap are not causing the slowdowns. Use a single heap size, if possible. Changing the heap size will require a restart of the Application Server. More information can be found in Chapter 8, “Java Virtual Machine” on page 191.
Examining the garbage collector. The JVM is frozen every time a full garbage collection is done. Enabling verbose garbage collection requires a restart of the application server. More information can be found in Chapter 8, “Java Virtual Machine” on page 191.

Work backwards from the last system used to the first used. This includes all subsystems.

Test the components separately, if possible, to locate which components have slow response times. This may require writing separate drivers or specific load testing to test the components.

Examine the application for performance bottlenecks. Many performance bottlenecks are caused problems in the application. More information can be found in Part 3, “Best practices - application” on page 267.

## 6.6 High CPU utilization

High CPU is usually caused by a loop in the application code. Processing like XML parsing is very heavy and could cause high CPU utilization.

Performance problems vary from one another; these are steps that can be taken to investigate why there is high CPU utilization:

- The software should be up to date. IBM releases fixes for WebSphere and the problem that is being experienced may have been corrected. Fixes, documentation, and other information can be found on the WebSphere Application Server for z/OS support page at:
  

- Examine the WebSphere and z/OS log files. The logs may provide information about the problem being experienced. More information can be found in Chapter 7, “WebSphere Application Server and HTTP Server” on page 159 and Chapter 9, “z/OS subsystems” on page 217.

- Take an SVC dump with maximized systrace. This provides the methods that the code has and where it came from, and could yield information about a loop. More information can be found in Chapter 9, “z/OS subsystems” on page 217.

- Take several thread dumps to determine which method the threads are executing. A single method could be causing the high CPU utilization. Several tools can be used to examine the thread dumps. More information can be found in Chapter 7, “WebSphere Application Server and HTTP Server” on page 159.

- Examine the garbage collector. The JVM is frozen every time a full garbage collection is done. Enabling verbose garbage collection requires a restart of
the Application Server. More information can be found in Chapter 8, “Java Virtual Machine” on page 191.

- Transactions, if being used, can be examined to identify which transaction is having unusually high times. The various components used in the transaction can then be examined to determine which specific component is utilizing CPU. More information can be found in Chapter 7, “WebSphere Application Server and HTTP Server” on page 159 and Chapter 9, “z/OS subsystems” on page 217.

- Make sure there is enough of a Java heap to run the application and that the expansion and contraction of the heap are not causing the slowdowns. Use a single heap size, if possible. Changing the heap size will require a restart of the Application Server. More information can be found in Chapter 8, “Java Virtual Machine” on page 191.

- Work backwards from the last system used to the first one used. This includes all subsystems.

- Test the components separately, if possible, to locate which components have slow response times. This may require writing separate drivers or specific load testing to test the components.

- Examine the application for performance bottlenecks. Many performance bottlenecks are caused problems in the application. More information can be found in Part 3, “Best practices - application” on page 267.

### 6.7 High memory utilization

Nearly all high memory issues deal with application problems. Objects are created and are not cleaned up in a timely fashion or finding place in the Java heap to place new objects causes memory fragmentation. It is always a good idea to have small objects and manually clean them up after use. New garbage collection technics, specifically generational garbage collection, can help with memory fragmentation, but at a price of some slower performance during garbage collection.
Performance problems vary from one another; these are steps that can be taken to investigate why there is high memory utilization:

- The software should be up to date. IBM releases fixes for WebSphere and the problem that is being experienced may have been corrected. Fixes, documentation, and other information can be found on the WebSphere Application Server for z/OS support page at:
  

- Examine the WebSphere and z/OS log files. The logs may provide information about the problem being experienced. More information can be found in Chapter 7, “WebSphere Application Server and HTTP Server” on page 159 and Chapter 9, “z/OS subsystems” on page 217.

- Take several thread dumps to determine which method the threads are executing. A single method could be causing the high memory utilization. Several tools can be used to examine the thread dumps. More information can be found in Chapter 7, “WebSphere Application Server and HTTP Server” on page 159.

- Examine the garbage collector. The JVM is frozen every time a full garbage collection is done. Enabling verbose garbage collection requires a restart of the application server. There are several different types of garbage collection. Different garbage collection types have different properties. A different garbage collector type may improve memory utilization. More information can be found in Chapter 8, “Java Virtual Machine” on page 191.

- Make sure there is enough of a Java heap to run the application and that the expansion and contraction of the heap are not causing the slowdowns. Use a single heap size, if possible. Changing the heap size will require a restart of the Application Server. More information can be found in Chapter 8, “Java Virtual Machine” on page 191.

- Take an SVC dump with maximized systrace. This provides the methods that the code is using and where it came from, and could yield information about a loop. More information can be found in Chapter 9, “z/OS subsystems” on page 217.

- Use the MODIFY commands in z/OS for WebSphere and examine their output. More information about which MODIFY commands are relevant are described in Chapter 7, “WebSphere Application Server and HTTP Server” on page 159.

- Workload activity reports provide a transaction response time analysis by MVS Workload Manager (WLM) service and report class. The reports are described in detail in Chapter 9, “z/OS subsystems” on page 217.

- Work backwards from the last system used to the first one used. This includes all subsystems.
Test the components separately, if possible, to locate which components have slow response times. This may require writing separate drivers or specific load testing to test the components.

Profile the application. Profiling will show how many objects are being created and their size in the Java heap. Several tools described in Chapter 5, “Setting up and using monitoring tools” on page 55 can help in profiling applications.

**Note:** The Problem Determination for WebSphere for z/OS series of IBM Redpapers can give more insight into debugging problems. These IBM Redpapers can be found at [http://www.redbooks.ibm.com](http://www.redbooks.ibm.com).
WebSphere Application Server and HTTP Server

In this chapter, we will discuss the IBM HTTP Server for z/OS and WebSphere Application Server for z/OS.
7.1 The WebSphere queuing network

WebSphere Application Server establishes a queuing network of interconnected queues that represent the various components of the application serving environment.

These queues, depicted in Figure 7-1, include the network, Web server (HTTP server), application server, and data source or connection to another back-end system. Each of these resources is represented by a queue of requests waiting to use that resource.

![Figure 7-1 WebSphere queuing network](image)

The first rule of tuning is to minimize the number of requests in WebSphere Application Server queues. In general, it is better for requests to wait in the network (in front of the Web server) than it is for them to wait in WebSphere Application Server. This configuration will result in allowing only requests that are ready to be processed into the queuing network. To accomplish this configuration, specify the queues furthest upstream (closest to the client) to be slightly larger, and specify the queues further downstream (furthest from the client) to be progressively smaller.

![Figure 7-2 UpStream queuing](image)
The queues in this queuing network, depicted in Figure 7-2 on page 160, are progressively smaller as work flows downstream. When 150 clients arrive at the Web server, 75 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 Application Server, 35 remain queued in the Web server and the remaining 40 are be handled by the Application Server. This process progresses through the data source until 15 users arrive at the final destination, the database server.

Because, at each point upstream, there is some work waiting to enter a component, no component in this system must wait for work to arrive. The bulk of the requests wait in the network, outside of WebSphere Application Server. This adds stability because no component is overloaded.

7.2 IBM HTTP Server and WebSphere Application Server plug-in

The IBM HTTP Server is the entry point into a WebSphere infrastructure for Web-based applications. The HTTP Server needs to be tuned to handle the expected workloads. When work is transferred to an application server, then this is performed using the WebSphere Plug-in, and additional tuning needs to be done here as well. This section examines the parameters that are available to tune these two components.

7.2.1 IBM HTTP Server

The first step in tuning the queuing network is to limit the amount of work that can be handled by the Web server. This is achieved by setting the \texttt{MaxActiveThreads} parameter.

\textbf{MaxActiveThreads}

The \texttt{MaxActiveThreads} directive is used to set the maximum number of threads that the HTTP server has active at one time to process incoming work. Each time your server receives a request from a client, it uses one or two threads to perform the requested action. One thread is used if the server is not performing DNS lookup; two threads, if the server is performing DNS lookup. If the maximum is reached, the server holds new requests until another request finishes and threads become available.
7.2.2 WebSphere plug-in

This section looks at the plug-in for IBM HTTP Server shipped as part of WebSphere Application Server Version 6 for z/OS. This function is called a plug-in because it runs as an extension of the IBM HTTP Server and interacts with the IBM HTTP server by means of the API interfaces provided by the IBM HTTP Server. Its main role is to direct traffic from the IBM HTTP Server to WebSphere Application Server instances and maintain "session affinity".

When workload arrives at the HTTP server, directives in its configuration file (httpd.conf) are used to determine who will handle the workload. The request can be handled by the HTTP server itself or passed on to the WebSphere Application Server Plug-in for processing.

Once control has been passed to the plug-in, it uses its own configuration file (plugin-cfg.xml) to determine how to pass the work on to WebSphere Application Server.

There are a number of settings that can be configured for the plug-in to improve its performance for the environment it is serving. The settings discussed here can be manipulated using the WebSphere Administrative Console and will be stored in the plug-in configuration file the next time it is regenerated.

**RefreshInterval**

The RefreshInterval property is the time interval (in seconds) at which the plug-in should check the configuration file to see if updates or changes have occurred. The plug-in checks the file for any modifications that have occurred since the last time the plug-in configuration was loaded.

**Recommendation:** The default value is set to 60 seconds, and this is ideal in a development environment where changes are frequent. However, in a production environment, a higher value is preferable, because updates to the configuration do not occur so often. A value of 600 seconds could be used as a sensible starting point.
If the plug-in reload fails for some reason, a message is written to the plug-in log file and the previous configuration is used until the plug-in configuration file successfully reloads.

To view the WebSphere Administrative Console page where you configure this property, click \texttt{Servers} \rightarrow \texttt{Web servers} \rightarrow \texttt{<webserver\_name>} \rightarrow \texttt{Plug-in properties}.

This page is shown in Figure 7-3 on page 164.

\textbf{LogLevel}

The \textit{LogLevel} property sets the detail of the log messages that the plug-in writes to the log file. You can specify one of the following four values for this property:

- \textbf{Trace}
  
  All of the steps in the request process are logged in detail.

- \textbf{Stats}
  
  The server selected for each request and other load balancing information relating to request handling is logged.

- \textbf{Warn}
  
  All warning and error messages resulting from abnormal request processing are logged.

- \textbf{Error}
  
  Only error messages resulting from abnormal request processing are logged.

\begin{tcolorbox}
\textbf{Recommendation:} The default value is "Error" and this is the recommended setting for production environments.
\end{tcolorbox}

You should check that this is the setting being used, as a setting previously used for problem investigation may not have been turned off. A "Trace" setting should never be used in a normally functioning environment, as it adversely affects performance.

To view the WebSphere Administrative console page where you configure this property, click \texttt{Servers} \rightarrow \texttt{Web servers} \rightarrow \texttt{<webserver\_name>} \rightarrow \texttt{Plug-in properties}. 
This page is shown in Figure 7-3.

Figure 7-3  Webserver plug-in properties

**RetryInterval**

The **RetryInterval** property specifies the length of time, in seconds, that the plug-in waits before trying to connect to a server that has been marked temporarily unavailable. The plug-in marks a server temporarily unavailable if the connection to the server fails.
Although the default value is 60 seconds, you might have to lower this value in order to increase throughput under heavy load conditions. Lowering the RetryInterval might help when the IBM HTTP Server is configured to have fewer than 10 threads per process.

How can lowering the RetryInterval affect throughput? If the plug-in attempts to connect to a particular application server while the application server threads are busy handling other connections, which happens under heavy load conditions, the connection might time out, causing the plug-in to mark the server temporarily unavailable. If the same plug-in process has other connections open to the same server and a response is received on one of these connections, the server is marked again. If there are only a few threads per IBM HTTP Server process, there might not be an established connection to this application server. When this situation occurs, the plug-in must wait for the entire retry interval.

**Note:** Although lowering the RetryInterval can improve performance if all the application servers are running, a low value can have an adverse affect when one of the application servers is down. In this case, each IBM HTTP Server process attempts to connect and fail more frequently, resulting in increased latency and decreased overall throughput.

To view the WebSphere Administrative Console page where you configure this property, select **Servers** → **Web servers** → `<webserver_name>` → **Plug-in properties** → **Request Routing**.

This page is shown in Figure 7-4 on page 166.

**LoadBalance**

The **LoadBalance** property specifies the load balancing option that the plug-in uses in sending requests to the application servers associated with that IBM HTTP Server.

The goal of the default load balance option, “Round Robin”, is to provide an even distribution of work across cluster members. Round Robin works best with Web servers that have a single process sending requests to the application server, for example, the HTTP server on z/OS. If the Web server is using multiple processes to send requests to the application server, the “Random” option can sometimes yield a more even distribution of work across the cluster.

**Recommendation:** Use a setting of Round Robin for HTTP Server on z/OS.
To view the WebSphere Administrative Console page where you configure this property, select **Servers → Web servers → <webserver_name> → Plug-in properties → Request Routing.**

This page is shown in Figure 7-4.

![Web servers Request routing](image)

**Figure 7-4  Web servers Request routing**

**ConnectTimeout**

The *ConnectTimeout* property enables the plug-in to perform non-blocking connections with the application server. Non-blocking connections are beneficial when the plug-in is unable to contact the destination to determine if the port is available or unavailable.

If no ConnectTimeout value is specified, the plug-in performs a blocking connect in which the plug-in sits until an operating system times out (as long as two minutes depending on the platform) and allows the plug-in to mark the server unavailable. A value of 0 also causes the plug-in to perform a blocking connect.
A value greater than 0 specifies the number of seconds you want the plug-in to wait for a successful connection. If a connection does not occur after that time interval, the plug-in marks the server unavailable and fails over to one of the other servers defined in the cluster.

**Recommendation:** A suggested value of 15, which means the plug-in waits for 15 seconds and if a connection does not occur after that time, the plug-in marks the server unavailable, is a good starting point for production environments.

To view the WebSphere Administrative console page where you configure this property, select Servers → Application servers → <server_name> → Additional Properties → Web server plug-in properties.

This page is shown in Figure 7-5 on page 168.

**MaxConnections**
The *MaxConnections* property is used to specify the maximum number of pending connections to an application server that can be flowing through a Web server process at any point in time.

By default, MaxConnections is set to -1. If this attribute is set to either zero or -1, there is no limit to the number of pending connections to the application servers.

During normal operation, the backlog of connections pending to an application server are bound to grow. Therefore, balancing workloads among application servers in a network fronted by a plug-in helps improve request response time.

It defines the maximum number of connections that can be pending to any of the application servers in the cluster. When this maximum number of connections is reached, the plug-in, when establishing connections, automatically skips that application server, and tries the next available application server. If no application servers are available, an HTTP 503 response code will be returned.

**Recommendation:** A suggested starting point is to use the Workload Profile Thread Count + 25%. This will maintain a steady flow of work to each server.

**Note:** This attribute is not necessary on the z/OS platform. The z/OS controller, working in conjunction with WLM, handles new connections dynamically.
To view the WebSphere Administrative console page where you configure this property, select **Servers → Application servers → <server_name> → Additional Properties → Web server plug-in properties**.

This page is shown in Figure 7-5.

![Figure 7-5 Webserver plug-in properties](image)

**LoadBalanceWeight**

The `LoadBalanceWeight` property is the weight associated with a server when the plug-in does weighted Round Robin load balancing.

The Round Robin load balancing implementation has a random starting point, that is, the first server will be picked randomly. Once started, Round Robin load balancing will use this weight to pick servers from that point forward.
The algorithm for this attribute decrements all weights within the server cluster until all weights reach zero. Once a particular server's weight reaches zero, no more requests are routed to that server until all servers in the cluster have a weight of zero. After all servers reach zero, the weights for all servers in the cluster are reset and the algorithm starts over.

This weighing can be used to direct more requests to servers that have greater resources available to them.

**Recommendation:** When a server is shut down for a period of time, we recommend that you set the weight for that server to zero. The plug-in can then reset the weights of the servers that are still running, and maintain proper load balancing.

To view the WebSphere Administrative console page where you configure this property, select **Servers → Clusters → <cluster_name> → Additional Properties → Cluster members.**

This page is shown in Figure 7-6.

![Cluster members](image)

**Figure 7-6  Cluster members**

### 7.2.3 References

- *WebSphere for z/OS Version 6 -- Understanding the HTTP Web Server Plugin*, PRS1467
- *z/OS HTTP Server Planning, Installing and Using*, SC34-4826
7.3 WebSphere Application Server

This section discusses WebSphere Application Server specific settings that could influence performance. Many of the settings depend on what the applications are doing and the amount and type of workload.

7.3.1 Tracing

Any tracing should be either disabled or minimized.

- In the WebSphere Admin Console, verify there is no JRAS tracing enabled by selecting Troubleshooting → Logs and Trace → <servername> → Change Log Detail Levels. The default is “*=info”, which includes informational messages and exception tracing, but you can also set it to “*=Off”, where there will be still some informational messages coming out.

- Verify the settings for the variables listed below. They are set under Environment → WebSphere Variables. Make sure you either look at the “All Scopes” scope view of the variables (V6.1 only) or look at the joblog output for the server in question. Since variables can be set at different levels or scope, you should make sure you are looking at the one that is currently in effect.

  **ras_trace_defaultTracingLevel=1 or 0**
  
  0 - no tracing at all.
  1 - exception tracing only.

  **ras_trace_basic** (not set)
  
  If the variable exists, delete it; you should only use this option when you are debugging a problem.

  **ras_trace_detail** (not set)
  
  If the variable exists, delete it; you should only use this option when you are debugging a problem.

  **ras_trace_outputLocation = BUFFER**
  
  The trace is written to a memory buffer and then written to CTRACE; make sure CTRACE is configured.

  **ras_trace_BufferCount=4**
  
  The Number of trace buffers to allocate.

  **ras_trace_BufferSize=128**
  
  The size of a single trace buffer in bytes. Generally, the larger the buffer allocation, the better the performance. However, specifying a buffer allocation that is too large can cause a decrease in performance due to system paging.

- Disable application tracing.
7.3.2 Workload Profile

Each servant address space has a limited number of worker (application) threads. Any application request that comes into the server will be processed using one of these worker threads. If there are no worker threads available (that is, all of them are busy executing), the requests will wait on the WLM queue. Depending on your settings for maximum servants (described in 7.3.3, “Controlling the number of servants” on page 172), a new servant may initialize to handle additional workload.

Workload Profile can be defined in administrative console by selecting the Servers → Application Servers → Container Services → ORB Service → z/OS additional settings → Workload Profile drop-down option. Table 7-1 shows possible options and when you would want to use them.

Table 7-1 Workload Profile options

<table>
<thead>
<tr>
<th>Workload Profile</th>
<th>When to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISOLATE</td>
<td>Number of Threads = 1</td>
</tr>
<tr>
<td></td>
<td>The servants are restricted to a single application thread. Use ISOLATE to ensure that concurrently dispatched applications do not run in the same servant. Two requests processed in the same servant can cause one request to corrupt another.</td>
</tr>
<tr>
<td>IOBOUND</td>
<td>Number of Threads = MIN(30,MAX(5,(Number of CPUs*3)))</td>
</tr>
<tr>
<td></td>
<td>Servants can have at a minimum of five application threads and a maximum of 30, where the exact number depends on the number of CPUs. IOBOUND is used by most applications that have a balance of CPU intensive and remote (I/O) operation calls. Example: A gateway or protocol converter application.</td>
</tr>
<tr>
<td>CPUBOUND</td>
<td>Number of Threads = MAX((Number of CPUs-1),3)</td>
</tr>
<tr>
<td></td>
<td>Servants can have at a minimum of three threads, where the exact number depends on the number of CPUs. Use CPUBOUND when an application performs processor-intensive operations, and therefore would not benefit from more threads than the number of CPUs. Example: An application with XML parsing and XML document construction - CPU intensive tasks.</td>
</tr>
<tr>
<td>LONGWAIT</td>
<td>Number of Threads = 40</td>
</tr>
<tr>
<td></td>
<td>The servants with 40 applications threads (more than any other setting). Use LONGWAIT when most of application processing is waiting for network or remote operations to complete. Example: An application makes frequent calls to another application system, like Customer Information Control System (CICS) window scraper applications, but does not do much of its own processing.</td>
</tr>
</tbody>
</table>
You can see the exact number of worker threads the server is running within the controller job output. Here’s a sample message:

BB0002341 SERVANT PROCESS THREAD COUNT IS 12.

If you are using a setting that is based on a CPU number, this number is determined based on the number of CPUs online when the controller comes up.

**Note:** The larger the number of threads per servant (JVM), the longer it takes to quiescent/suspend and restart the threads in order for Garbage Collection (GC) to be able to run. It also means that not all threads are available to continue with work during the GC cycle.

### 7.3.3 Controlling the number of servants

There are different ways of controlling the number of servants running and available to accept work depending on a situation. The two main settings that control how many servants the application server is able to run with are located under **Servers → Application Servers → <serverName> → Server Infrastructure → Java and Process Management → Server Instance.**

Make sure you have the “Multiple Instances Enabled” check box checked.

**Minimum Number of Instances**

(wlm_minimumSRCount in joblog output) Specifies the minimum number of servant process instances to activate. This is the number of servants that will always be up and running regardless of what the workload is. It is often a good idea to have this number set higher than 1 (one). If for some reason the servant terminates (for example, a transaction timeout that brings down the servant with EC3/0413000x abend), you still have other servants ready to accept work while the other one is terminating and a new one is initializing. If you expect a higher workload that will require affinity (for example, session objects), you may also consider setting this variable to a higher number along with the **WLMStatefulSession** variable. This situation is described in more detail in 9.3.2, “Temporal affinity workload balancing” on page 227.
Maximum Number of Instances

(wlm_maximumSRCount in jolog output) Specifies the maximum number of servant process instances to activate. Setting this variable to 0 (zero) will allow WLM to start up as many servants as needed. Setting it to anything higher than zero will prevent WLM from starting any more servants than that defined number, regardless of how high the workload is.

You will also want to review the above settings if your server is configured to use different WLM transaction classes. For more information about this topic, see 9.3.1, “WLM classification” on page 222.

7.3.4 Timeouts effecting performance

WebSphere Application Server on z/OS does not have ability to terminate one thread for the request that timed out and ensure that all the resources are freed. The only way to do this at this time is to abend the servant region with abend EC3/0413000x. This section includes some things you can do to minimize the effect of timeouts. Of course, regardless of how WebSphere handles timeouts, it is still a timeout, which ultimately means the client will not get the expected response. A timeout should be investigated to determine the reason why this condition occurred in the first place. For step-by-step instructions on how to approach this problem, you may want to look at Chapter 7, “Timeout”, in Problem Determination for WebSphere for z/OS, SG24-6880.

There are two ways to avoid EC3/0413000x abends for certain timeout. However, before using either of them, you need to review the side effects they may cause that could be more severe than the occasional EC3 abend.

- protocol_http_timeout_output_recovery: This variable affects HTTP timeouts only. Other timeouts (for example, WLM or transaction timeouts) will not be affected by it. By default, this variable is set to SERVANT. When an HTTP timeout is encountered, the servant will be abended with an EC3/04130007 abend. If you set this variable to SESSION, the servant will not be terminated. The request will be allowed to continue running as long as it wants to. The only thing that will be done is the HTTP session and socket clean up in the controller. The client will be notified that the timeout occurred. If the request does happen to finish after the timeout occurred, the transaction will be rolled back and the thread will be available for new requests. If it does not complete (for example, there is a deadlock), the thread will be in this state until the servant is recycled. This means the thread will not be available for future requests. In addition, any resources held by this thread will not be released, which could cause some other problems. This could also lead to a slower performance or no response from the application server if multiple or even all
threads are in this hung condition. You can set this variable in the WebSphere Admin Console by selecting **Servers → Application Servers →** <serverName> → **Server infrastructure → Administration → Custom Properties**.

- Disable timeouts (set them to 0 - zero): The timeout that is set to zero will not be activated when requests comes in. The behavior will be similar to setting the `protocol_http_timeout_output_recovery` variable to **SESSION** with a couple of differences. First, any timeout can be set to zero, not just HTTP timeout. Second, since there is no timeout, the controller will not take any actions. Look in the InfoCenter for names of the specific timers you want to disable.

**Warning:** Do not disable timeouts or set `protocol_http_timeout_output_recovery = SESSION` unless it is absolutely necessary and all other solutions failed. Review the rest of this section for possible alternatives in different situations.

**Bouncing server effect**

When in production (with a constant load of incoming requests), one of the following two situations could occur, causing a constant restart of the server due to timeouts (bouncing server), which extends the server downtime:

1. The application server (controller and servants) needs to be either restarted or it terminates for different reasons and a new one has to initialize.

   The controller will initialize first and the listener will very likely be started before your minimum number of servant instances are initialized and ready to handle the incoming workload. Many of these requests will be waiting on the WLM queue with the timer already running. By the time they are dispatched to a servant, they might not have enough time to finish processing and a timeout (that causes another servant termination) will be issued.

   **Solution:** Set `protocol_accept_http_work_after_min_srs = 1`.

   When this variable is set to 1 (true), the application server controller does not accept any HTTP work requests (that is, the HTTP listeners are disabled) until the minimum number of servants (configured with `wlm_minimumSRCount`) are initialized and ready to accept work.
2. The application server servant (not controller) terminates abnormally.

The most common reason for servant abnormal termination is when a timeout is hit (abend EC3/0413000x) or a JVM OutOfMemory condition is encountered. If the server was running with only one servant instance or the rest of the servants are not able to handle the amount of incoming requests, the requests will start backing up on the WLM queue and by the time the new servant is initialized, the request might start timing out causing another servant termination.

Solution: Set control_region_dreg_on_no_srs = 1.

When this variable is set to 1 (true), the application server controller will stop accepting HTTP work requests (the HTTP listeners will be stopped) until the minimum number of servant instances (configured with wlm_minimumSRCount) are initialized and ready to accept work.

You may want to consider setting the following variable, which works in conjunction with the variable above. If it is set to 1 (true), a WTOR message will be issued and the listeners will remain disabled until response to the message is provided:

control_region_confirm_recovery_on_no_srs=1

The above variables can be set by selecting Application servers → serverName → Server infrastructure → Administration → Custom Properties.

These variables provide the ability to disable listeners during the time the server is not ready to process requests. This also means, that from the client perspective, the server will appear unavailable during the time the listeners are disabled.

**Timeout grace period**

When one request hits a timeout, the servant is recycled (abended with EC3/0413000x abend and a new one is restarted). That means that all other requests that were also executing in that servant would also be terminated. It might be that this one request was the only long running request and the other requests fell victim to this one. There is a variable now that causes the servant to delay the abend, allowing the other threads to complete:

control_region_timeout_delay = value in seconds

You can set this variable in admin console by selecting Servers → Application Servers → <servername> → Server Infrastructure → Administration → Custom Properties.
The default for this variable is 0 - zero. If it is set to a value greater than zero, the next time a timeout is hit, the servant will continue processing existing requests for the length of time specified in the delay variable. If any of the other requests do complete, the freed up threads will be put to sleep (that is, they will not be able to pick up new work). Once the time specified in the delay variable passes, the servant will be abended with an EC3/0413000x abend as usual.

7.3.5 Security

Enabling security adds additional overhead for each transaction. While having security enabled in your environment might be required, you might be running with more than you need. Consider these points to see if you can reduce the amount of overhead caused by security:

- Use the minimum number of EJBROLES on methods. If you are using EJBROLES, specifying more roles on a method will lead to more access checks that need to be executed and a slower overall method dispatch. If you are not using EJBROLES, do not activate the class.
- Disable Java 2 Security if you do not need it. It is used to restrict application access to local resources. When you enable security, Java 2 Security is enabled by default. Verify the setting by going to WebSphere Administrative Console and selecting Security → Secure administration, applications, and infrastructure and finding the “Java 2 security” box on this page. See Figure 7-7.

![Figure 7-7  Java 2 Security option](image)

- Use the lowest level of authorization consistent with your security needs. You have the following options when dealing with authentication:

  **Local authentication**
  
  Local authentication is the fastest type because it is highly optimized.

  **UserID and password authentication**
  
  Authentication that utilizes a user ID and password has a high first-call cost and a lower cost with each subsequent call.

  **Kerberos security authentication**
SSL security authentication

SSL security is notorious in the industry for its performance overhead. Luckily, there is a lot of assistance available from the hardware to make this reasonable on z/OS.

- If using Secure Sockets Layer (SSL), select the lowest level of encryption consistent with your security requirements. WebSphere Application Server enables you to select which cipher suites you use. The cipher suites dictate the encryption strength of the connection. The higher the encryption strength, the greater the impact on performance.

- Disable Security Attribute Propagation (SAP®). In V6, it is enabled by default. It is used to send security attribute information regarding the original login to other servers using a token (used with SSO - Single Sign-On). To completely disable SAP, in the administrative console, go to Security → Global Security →:
  - Authentication mechanisms → LTPA or ICSF → Single sign-on (SSO) and uncheck Web inbound security attribute propagation.
  - Authentication protocol → CSIV2 inbound authentication and uncheck Security attribute propagation.
  - Authentication protocol → CSIV2 outbound authentication and uncheck Security attribute propagation.

You can read more about SAP in the InfoCenter at:


7.3.6 Dynamic Cache Service

Dynamic Cache Service is used to improve application performance by caching the output of servlets, commands, and JavaServer Pages (JSP) into memory. In order to use this service, you need to develop the cache policy file, cachespec.xml. It allows you to specify what you want to cache and how long you want to keep it. You can find more information about how to create cachespec.xml in the WebSphere Application Server InfoCenter. Here’s a simple example:

```xml
<cache-entry>
  <class>servlet</class>
  <name>/servlet/CommandProcessor</name>
  <cache-id>
    <idgenerator>com.mycompany.SampleIdGeneratorImpl</idgenerator>
    <timeout>60</timeout>
  </cache-id>
</cache-entry>
```
To enable dynamic caching globally, go to the WebSphere Admin Console and select **Servers → Application Servers → Container Services → Dynamic Cache Service**. See the screen capture in Figure 7-8.

![Application servers > pfsrc01a > Dynamic cache service](image)

**Figure 7-8  Enabling Dynamic Cache Service**

You will need to specify what type of caching you want to enable:

**Servlet caching**  
Select **Servers → Application servers → server_name → Web container settings → Web container** and check the check box next to **Enable servlet caching**.

**Portlet fragment caching**  
Select **Servers → Application servers → server_name → Portlet container settings → Portlet container** and check the check box next to **Enable servlet caching**.

**Edge side include caching**  
This is configured through the plugin-cfg.xml file.

**Command caching**  
You need to define an interface.

**Web services client cache**  
You need to defined the cache policy file based on Web Services Description Language (WSDL) file for the remote service.
7.3.7 LogStream compression

If your application components are using recovery log services (for example, Transaction Service - XA partner log and the Compensation Service components), there is a WebSphere variable you may need to take a look at that indicates how frequently the recovery log service attempts to compress the LogStreams. The default is 30 seconds. Consider disabling the compression (set the variable to 0 - zero) if you are not using these services or set it high. The LogStream is checked for compression once per specified interval. This operation can cause unnecessary CPU usage in the controller if the LogStream is not being used. However, if your recovery log service uses logstreams, do not set this property to too high a value. If the recovery log service logstreams become full before the compression interval expires, transactions might start to fail until the logstreams are compressed.

You can set this variable in the WebSphere Admin Console by selecting
Servers → Application Servers → <serverName> → Container Services → Transaction service → Custom Properties and add the following variable:

RLS_LOGSTREAM_COMPRESS_INTERVAL = value in seconds

This function was shipped in 6.0.2.1 with apar PK08027.

You may also consider using HFS for the XA partner log. To configure it, follow this path in the WebSphere Admin Console: Servers → Application Servers → <serverName> → Container Services → Transaction Service.

7.3.8 MonitoringPolicy Ping interval

In a Network Deployment environment, the Node Agent will attempt to contact application servers to make sure they are still there. How often Node Agent sends the isAlive SOAP messages is controlled by the MonitoringPolicy Ping interval. Setting this value higher will reduce CPU usage in the controller. The default setting is 60 seconds.

To change the setting, select Servers → Application Servers → <serverName> → Java and Process Management → Process Definition → Control → MonitoringPolicy → Ping interval.

Decreasing the value detects failures sooner; increasing the value reduces the frequency of pings, reducing system overhead.
7.3.9 File synchronization

Every time you save changes you make in the admin console, you have an option to synchronize the changes across multiple nodes. In addition, there is an automatic synchronization that is enabled by default. You may want to increase the synchronization interval depending on your requirements. The default is 10 minutes. Frequent synchronizations on the system with little changes can produce unnecessary overhead.

You can find the synchronization interval setting in the Admin Console under System Administration → Node Agents → node_agent_name → File Synchronization service.

On a production system where configuration updates are infrequent, you may want to consider disabling automatic file synchronization. To do this, uncheck the Automatic synchronization check box. If you are running your environment with SSL encryption enabled, you still need to do some synchronization to exchange newly generated keys (they will be generated every 90 days per default). This can be achieved by enabling the "Startup synchronization" parameter if the system is regularly restarted or by disabling the generation of new keys by setting the following parameters:

- Dynamically update the run time when SSL configuration changes occur (select Security → SSL certificate and key management).
- Automatically replace expiring self-signed certificates (Select Security → SSL certificate and key management → Manage certificate expiration).
- Automatically generate keys (Select Security → Secure administration, applications, and infrastructure → Authentication mechanisms and expiration → Key set groups → CellLTPAKeySetGroup).

**Note:** Ping Interval is needed to recognize when an application server failed and needs to be restarted. However, the restart of the server will only occur when Automatic Restart, located on the same page, is set to true. On z/OS, the default is false (disabled).
Make sure both application servers and node agent are restarted after this change. See Figure 7-9 for a screen capture of the admin console page with these settings.

Figure 7-9 File synchronization service
7.3.10 Object Request Broker (ORB)

If you have an EJB client and the EJB in the same classloader (deployed in the same EAR file), configure the Object Request Broker (ORB) so it passes parameters by references instead of by value. It is expensive to copy large objects (non-primitive) and performance is improved greatly when pass by reference is used. You can enable pass by reference in the Admin Console by selecting Servers → Application servers → server_name → Container services → ORB service. See the screen capture in Figure 7-10. This option is equivalent to com.ibm.CORBA.iiop.noLocalCopies.

Another option to look at is com.ibm.CORBA.FragmentSize. If you expect larger ORB General Inter-ORB Protocol (GIOP) messages, over 1024 bytes, which is the default, you may want to increase this value. This variable specifies the size of the fragments used by ORB. If the request is larger than the specified size, the ORB will break up the request into fragments before sending it.
7.3.11 High Availability (HA) Manager

An HA Manager provides several features that allow other WebSphere Application Server components to make themselves highly available (for example, the “bulletin board” mechanism, which is a framework for high speed and reliable messaging between processes). However, it also consumes valuable system resources, such as CPU cycles, heap memory, and sockets. HA Manager can be disabled per process (that is, per application server). Consider disabling it for a process that does not use any of the following services:

**Note:** You should not disable HA Manager on a Node Agent or Deployment Manager unless it is also disabled on all servers that belong to that core group.

- Memory-to-memory replication. If you have one of the following configured, then you are using memory-to-memory replication:
  - Memory-to-memory replication is enabled for Web container HTTP sessions.
  - Cache replication is enabled for the dynamic cache service.
  - EJB stateful session bean failover is enabled for an application server.
- Singleton failover, cluster-based service.
- Workload management routing. This is not the same as z/OS WLM. Generally used on distributed WebSphere Application Servers.
- On-demand configuration routing, used for both proxy server and Web services routing.

At the time of writing of this book, disabling HA Manager is not possible through the Admin Console, though there are plans to make this possible. You can only disable it through the `wsadmin` tool. A sample script is available at the following Web site:

7.3.12 Hot Deployment and Dynamic Application Reloading

*Hot Deployment* was designed to be able to quickly replace a specific module of the application (for example, an EAR, WAR, or JAR file) while the application is started and currently in use. The change to the module is recognized by checking for changes to the files in the HFS. It is advisable to disable *Dynamic Application Reloading* in a production environment for two reasons. First, it poses unacceptable risks to production environments. Full or partial resynchronization might erase hot deployed components. Second, HFS access is very expensive. Even if there are no changes to the application, the check to see if files were updated is still done. The frequency of this check depends on the reloading interval value.

In order to disable Dynamic Application Reloading and therefore reduce HFS access, you need to:

- Disable EJB and WAR modules reloading (not JSPs) using one of the following methods:
  - Through the application development tooling (for example, RAD) in the ibm-web-ext.xmi file. Many toolings set the reloadInterval low (for example, three seconds):
    
    ```
    rloadInterval=”0” reloadingEnabled=”false”
    ```
  - In the Admin Console, during deployment. See Figure 7-11 on page 185.
  - In the Admin Console, once the application is already deployed, by selecting **Applications** → **Enterprise Applications** → `<appName>` → **Class loading** and update detection.
Disable JSPs reloading using one of the following methods:

- Through the application development tooling (for example, RAD) in the `ibm-web-ext.xmi` file:

```xml
<jspAttributes xmi:id="JSPAttribute_1" name="reloadEnabled" value="false"/>
<jspAttributes xmi:id="JSPAttribute_2" name="reloadInterval" value="0"/>
```
In the Admin Console, during application deployment. See Figure 7-12.

In the Admin Console, once the application is already deployed, by selection **Applications → Enterprise Applications → <appName> → JSP reload options for web modules**.

If you have the following set, JSP reloading will be disabled automatically:

```xml
<jspAttributes xmi:id="JSPAttribute_1" name="disableJspRuntimeCompilation" value="true"/>
```

Note, you need to pre-compile JSPs first. See 7.3.13, “Web container” on page 186 for more details on pre-compiling JSPs.

**Figure 7-12  JSP reloading option during application deployment**

### 7.3.13 Web container

The following sections contain some best practices that specifically apply to the Web container.

**Pre-compile JavaServer Pages (JSPs)**

If you do not pre-compile JSPs in your application, the first invocation of JSPs will always take longer to complete. Pre-compiling JSPs will also reduce compilation related disk operations. You can pre-compile JSPs at different points of the deployment process:

- In Rational Application Developer (RAD) or other development tools.
In the Admin Console when deploying an application. One of the windows titled Select installation options will have a check box with the description Precompile JavaServer Pages files. By default, this check box is not checked.

Once the application is deployed, you can still run a JSP batch compiler tool. You can read more about this tool in the WebSphere InfoCenter.

If for some reason you are not able to pre-compile JSPs, consider setting jspCompileClasspath, which became available in WebSphere Application Server V6. With this option set, the application will use what is referred to as a “short” or “small” classpath as opposed to the often lengthy WebSphere CLASSPATH. WebSphere’s CLASSPATH includes many WebSphere Application Server specific classes making the CLASSPATH long. During each JSP compile, the search for a jar is done looking through the whole classpath starting from the beginning until the required jar is found. Since WebSphere specific jar files are first, they are often searched unnecessarily. The short classpath includes only a subset of the jar files and excludes many of the WebSphere jars.

You can set jspCompileClasspath in WEB-INF/ibm-web-ext.xmi with the following line:

```
<jspAttributes xmi:id="JSPAttribute_8" name="jspCompileClasspath" value=""/>
```

If you do not specify any value, the JSP engine will only use the short classpath. If your application requires you to use certain files that are not included in the short classpath (for example, WebSphere public APIs), you can specify them in the value field.

**URL invocation cache**

The **URL invocation cache** holds information for mapping request URLs to servlet resources. A cache of the requested size is created for each worker thread that is available to process a request. The default size of the invocation cache is 50. If more than 50 unique URLs are actively being used (each JavaServer Page is a unique URL), you should increase the size of the invocation cache.

**Note:** A larger cache uses more of the Java heap, so you might also need to increase the maximum Java heap size. For example, if each cache entry requires 2 KB, the maximum thread size is set to 25, and the URL invocation cache size is 100; then 5 MB of Java heap is required.
To set the invocation cache, follow this path in the Admin Console: **Servers → Application Servers → <servername> → Java and Process Management → Process Definition → Control or Servant → Java Virtual Machine → Custom Properties** and add the following property:

```
invocationCacheSize = integer number
```

**Session management**
To manage the parameters of your session, select **Servers → Application servers → <servername> → Web container settings → Web Container → Session management → Distributed environment settings → Custom tuning parameters**, which will produce the window shown in Figure 7-13.

![Figure 7-13  Session Management tuning options.](image)
7.3.14 EJB container

Tune the *EJB container cache size and cleanup interval*. You can find the settings for these items in the Admin Console by selecting **Servers → Application Servers → serverName → EJB Container Settings → EJB cache settings**.

The cache size specifies the number of buckets in the active instance list within the EJB container. To maximize performance, each bucket in the table should have a minimum number of instances assigned to it. However, for the best balance of performance and memory, set this value to the maximum number of active instances expected during a typical workload.

The cleanup interval (the default is 3000 milliseconds) specifies how often unused cache items are removed. In general, increase this parameter as the cache size increases.
Java Virtual Machine

As mentioned earlier in this book, the Java Virtual Machine is a key component in WebSphere Application Server and is one of the first places to look in case of performance problems. Also, the JVM settings should be investigated thoroughly before deploying any new application, in conjunction with the workload projections for that application and the application characteristics.

The focus of this chapter is on the combination of the JVM heap and the Garbage Collector (GC). We also show how the output of “verbose GC” can be used to determine the behavior and thus the bottlenecks of the JVM, or actually the bottlenecks produced by the application.

We conclude the chapter with a few notes on the Just-In-Time (JIT) compiler.
8.1 Introduction

Tuning the Java Virtual Machine (JVM) is not always easy and it is sometimes difficult to discover where to start. Before we give you some best practices for verifying the health of your WebSphere environment and the applications that are running in it, we will explain some details about the JVM.

The JVM is an important part of the Java Runtime Environment (JRE) that resides in the WebSphere Application Server. Besides the JVM, the JRE consists of the Just In Time (JIT) compiler, Java platform core classes and supporting files, and a Garbage Collector (GC).

The high-level structure of the JRE is shown in Figure 8-1.

![Figure 8-1 Components of a Java application stack and the IBM JRE](image-url)
8.2  JVM heap and Garbage Collection

Here we discuss the JVM heap and Garbage Collection.

8.2.1  JVM storage allocation

The heap is the storage of the JVM. The heap is managed by the Garbage Collector, which allocates memory blocks and collects unused memory blocks (so called Garbage Collection). In Java, these memory blocks are called objects. An object stays alive as long as it is reachable. This means that if an object is not referenced anymore, it becomes “garbage”.

Allocation of an object will normally cause a heap lock, to prevent other threads from updating simultaneously. To avoid too many heap locks, every thread has a local allocation buffer in which it will allocate all objects less than 512 bytes (or 768 on 64-bit platforms). Larger objects will be allocated in this cache also if they fit in the remaining space of the cache. As the local cache becomes full, a heap lock has to be issued to obtain a new cache buffer.

z/OS allocation

The default maximum heap size in a z/OS environment is 64 MB. With the -Xmx option, this size can be changed. The initial heap size in z/OS is 1 MB and can be altered by the -Xms parameter. At JVM startup time, the specified (or default) maximum heap size will be allocated in z/OS. Only the initial heap size will be used in the JVM until the heap is expanded by the Garbage Collector after an allocation failure.

Heap expansion

If the maximum heap size is not reached, there are three events that will cause the Garbage Collector to expand the heap:

- If the Garbage Collector could not free enough space to satisfy the allocation request.
- If the free space in the heap is less than the minimum specified in the -Xminf parameter. The default is 30%.
- If the available free space goes below a minimum level. The default is 13% and this value can be changed by the parameter -Xmaxt.
Heap shrinkage
Heap shrinkage will occur if these conditions are true:

- The Garbage Collector frees enough space to satisfy the allocation request.
- The maximum free space parameter (-Xmaxf) is not set to 100%.
- The heap has not been expanded during the last three Garbage Collection cycles.

During shrinking, the storage is released in the virtual space of the JVM. In z/OS, the Garbage Collector uses commit/decommit to release physical storage.

Setting up heap size
To change the parameters to affect the heap size, you have to log on to the Admin Console and follow this flow:

2. There are boxes to fill in the Initial heap size (in MB) and the Maximum heap size.
3. In the box “Generic JVM arguments”, you can add other JVM parameters, such as -Xminf, -Xmaxf, -Xmmine, -Xmaxe, and so on.

To activate the new settings, you have to restart the server.

Although the most likely place to change the heap parameters is the one of the servant regions, you can also change the heap size of the control or the Adjunct region to fine-tune your storage usage.

The best way to do this is observing the used heap by these regions by analyzing the VerboseGC for a period of time.

Different areas in the heap
At startup time, the heap is divided into two areas, the Large Object Area (LOA) for objects with a size of 64 KB or more, and a Small Object Area (SOA). This will reduce fragmentation of the heap. The size of the LOA is determined at initialization time and recalculated during every Garbage Collection, based on free space in the LOA and the SOA. The size of the LOA can be adjusted with the following properties:

- Xloainitial for the initial size of the LOA.
- Xloamaximum for the maximum size of the LOA.
Allocation of objects will be done first in the SOA. If there is not enough continuous space, a retry is done in the LOA if the object is 64 KB or bigger. If it is smaller, an allocation failure occurs and a Garbage Collection will be started.

To disable the creation of a LOA, you can specify -Xnoloa.

8.2.2 Garbage Collection policy

There are four different Garbage Collection (GC) policies to clean up your heap.

▷ optthruput
   This option is the default value and delivers a very high throughput to your application. A disadvantage can be occasional pauses. With this policy, concurrent mark is disabled and your application has to wait until the mark phase is completed.

▷ optavgpause
   This option gives slightly more overhead, but it reduces the time spent in Garbage Collection pauses. It enables concurrent Mark and should be considered for very large heaps.

▷ gencon
   With this policy, the heap is split up in two areas, one for new allocations and one for objects that have stayed in the heap for longer time. It works with a combination of concurrent and generational Garbage Collection.

▷ subpool
   This policy uses an improved allocation algorithm for objects. During our test, we were not able to get this policy working, probably because we were testing with beta code.

Depending on the behavior of your application and the level of service that is required, you have to choose one of these policies.
8.2.3 Garbage Collection

When a request for storage cannot be accomplished because there is not enough contiguous space, an allocation failure occurs. This triggers the Garbage Collector. The Garbage Collector stops all the work and starts the Garbage Collection. This is done in two to three phases:

- **Mark**: In this phase, all thread stacks are scanned to find references to Java objects. These Java objects are scanned in the second part of the mark phase for references to other Java objects. The results of this scan are stored in a bit vector.

- **Sweep**: The bit vector from the mark phase will be used here to find the garbage. This garbage is added to the pool of free space.

- **Compaction (only when necessary)**

Figure 8-2 shows an example of the heap before any compaction has taken place. The objects are basically scattered around, and not necessarily using continuous space. Figure 8-3 on page 197 shows an example of the heap after a compaction has taken place.

![Figure 8-2 Compaction of the heap 1: before compaction](image)
8.3 Analyzing VerboseGC

In the following sections, we will show how the output of the Garbage Collector can be used to analyze the behavior of the Garbage Collector.

8.3.1 Introduction

An inexpensive and effective method to analyze the behavior of your JVM is switching on the VerboseGC log. It is inexpensive because it does not take much CPU cycles to have it always on, even in a production environment.

The best thing to do is to redirect the output to a file, so the contents can be processed by an application or tool later.

Switching on the VerboseGC has to be done via the WebSphere Admin Console. After changing this value, you have to recycle the servant region.

The GC log contains detailed information about every Garbage Collection event; however, the flood of information that comes from the JVM is hard to understand. We tried to use different tools to make the information clear, but none of the tools
were able to process the new XML formatted VerboseGC log from the SDK 5.0, which comes with WebSphere Application Server Version 6.1.

We were able to make a little program by ourselves to do the job for us during the time we worked on this book. We called it GC Analyzer.

Figure 8-4 and Figure 8-5 show examples of the GC log using different policies. Those are the logs that we used with our tool.
8.3.2 Our GC Analyzer tool

To emphasize the workings of the Garbage Collection and to give you some hints and tips, we like to show you a few examples of the output of this program to look at when you are analyzing your own VerboseGC output.

The GC Analyzer program we created is part of the additional material for this book. See Appendix B, “Additional material” on page 395 for downloading instructions.

The GC Analyzer program reads the VerboseGC output from the JVM 1.5 on z/OS and creates several charts in an HTML file. It also creates a delimiter (tab) separated file from where you can easily make your own charts with any spreadsheet program you prefer.

The default charts that will come with the program are:
- GC interval, discussed in “GC interval” on page 200
- GC time, discussed in “GC Time” on page 202
- Percent of elapsed time spent in GC, discussed in “Percent of elapsed time spent in GC” on page 203
- Mark time, discussed in “Mark time” on page 205
- Sweep time, discussed in “Sweep time” on page 206
- Compact time, discussed in “Compact time” on page 206
- Combined Mark, Sweep, and Compact time, discussed in “Combined Mark, Sweep, and Compact time” on page 208
- Nursery free bytes, discussed in “Nursery free bytes” on page 209
- Tenured free bytes, discussed in “Tenured free bytes” on page 210
- LOA bytes free, discussed in “LOA bytes free” on page 211
- SOA bytes free, discussed in “SOA bytes free” on page 212
- Requested bytes, discussed in “Requested bytes” on page 213
- Finalized objects on queue, discussed in “Finalized objects on queue” on page 213

Depending on the selected GCPolicy (see 8.2.2, “Garbage Collection policy” on page 195), the VerboseGC output will contain specific entries. The GC Analyzer will produce an empty chart if there is no information in the VerboseGC log.
8.3.3 GC Analyzer charts

Here you will find the set of charts produced by our GC Analyzer tool. We have used different VerboseGC log files for the individual charts to make them more clear.

GC interval

Figure 8-6 shows a chart of the time between the successive Garbage Collections. This implies that our focus is not on the high values, but on the lowest values.

For example, if there is no activity during the night, the interval value will be high, but if there is a lot of Garbage Collection activity, the values will be low.
Unfortunately, the scale of this chart does not show the low values in detail. To get a better view of the area where we have to put attention to, we would have to make another chart with a spreadsheet application, as shown in Figure 8-7.

By limiting the height of the vertical scale, as shown in Figure 8-7, more details about Garbage Collection intervals below 30 seconds are shown. Times of more than 300 seconds are just fine. The values below 30 seconds in this chart need attention. At several moments in time, you will find a lot of Garbage Collection Cycles every 5 to 10 seconds. A short time between Garbage Collections might be a problem, but it does not have to be. It also depends on the time that is needed for each Garbage Collection.
**GC Time**

Figure 8-8 shows the time spent with Garbage Collection. The higher values need our attention, because during the Garbage Collection the world stands still. A Garbage Collection time of more than one second can be noticed by the users, especially when the frequency of the Garbage Collection is high. In our case, we saw a frequency of the Garbage Collection every 5 to 10 seconds.

![GC Time Chart](image)

*Figure 8-8  GC Time*

Frequency is another thing to look at in this perspective, as well as the time of the day. If you are running batch processes on your system during off peak hours, the WebSphere Servant task can be paged out and the first user can run into a paging activity that can take tens of seconds.

Also, CPU intense events like producing a dump can cause a sudden spike in your chart. To get a reasonable view of the overhead caused by Garbage Collection, a combined chart has to be made.
Percent of elapsed time spent in GC
Based on the total elapsed time, this chart gives you an idea about which part of the total time is spent with Garbage Collection. The first chart (Figure 8-9) shows a healthy Garbage Collection. In our opinion, this value should not rise above 5%.

Figure 8-9  Percentage of elapsed time spent in Garbage Collection (good)
In the second chart (Figure 8-10), the percentage is far above the 5%, so the Garbage Collection needs your attention. With the next charts, we will drill down into more detail and will be able to point out the cause of the poor performance of the Garbage Collection and your JVM.

Figure 8-10  Percentage of elapsed time spent in Garbage Collection (bad)
**Mark time**

As we have described in 8.2.3, “Garbage Collection” on page 196, during the Mark process, all reachable piece of memory (Java Objects) will be identified. During our tests, we did not observe any strange behavior in the Mark phase. See Figure 8-11.

![Mark Interval Graph](image)

*Figure 8-11  Mark time*
Sweep time
During the Sweep phase, all memory locations that are not reachable and are not marked in the Mark phase are added to the pool of free space (see 8.2.3, “Garbage Collection” on page 196). During our tests, we did not observe any strange behavior in the Sweep phase. See Figure 8-12.

![Sweep Interval](image)

**Figure 8-12** Sweep time

Compact time
The Compact phase is different from the Mark and Sweep phase. The Garbage Collector will only compact the heap when it is necessary, because it is an expensive process. If the Sweep phase provides enough continuous free space to fulfill the original request, the Compaction phase will not occur.
As you can see in the compaction phase of our test (Figure 8-13), the duration of the Compaction phase is quite large. The heapsize of the JVM is probably too small for this type of work or the application allocates big objects.

Figure 8-13  Compact time
Combined Mark, Sweep, and Compact time
This combined chart, shown in Figure 8-14, gives an idea of the time spent for the three different phases in the Garbage collection. In our example, the time spent for the compaction is much more than for Mark or Sweep, but in your environment, you may see different figures. The user of the application in this sample will observe a fluctuating response time.

![Mark, Sweep, Compact Interval](image)

*Figure 8-14 Combined Mark, Sweep and Compaction time*

The conclusion, after a look at this chart, should be:

Check if your heap is big enough. You can do this by checking the value of percentage free after GC. Depending on the Garbage Collection Policy chosen for this JVM, you have to look at the Tenured, Nursery, SOA, or LOA values.

If there is enough free space left, then you probably have to take a look at the allocation behavior of your application.
Nursery free bytes
The chart shown in Figure 8-15 is produced with the log info of a JVM with policy GENCON. As you can see, almost the whole nursery area is freed during a Garbage Collection cycle.

As you can see in this chart, the amount of free bytes in the nursery area is growing. This storage is borrowed from the tenured area (see Figure 8-16 on page 210), which obviously does not need this space. The distribution of memory between the nursery and the tenured area is determined dynamically. The percentage can be found in the Garbage Collection Log and is named scavenger tiltratio.

Figure 8-15  Nursery free bytes
**Tenured free bytes**

Figure 8-15 on page 209 shows a growth of the nursery area; in Figure 8-16, the size of the nursery area is shrinking. Probably many short living objects are allocated and they do not live long enough to become a candidate for the tenured area.

If this happens in your production environment, you should consider a review of your application.

*Figure 8-16  Tenured free bytes*
LOA bytes free
In Figure 8-17, you can see the shrinkage of the LOA. The number of bytes free is almost as high as the total number of bytes.

Figure 8-17  LOA bytes free
SOA bytes free
The expansion of the tenured area has also effect on the size of the SOA, as shown in Figure 8-18. See the growth of the SOA at 20:45.

![SOA Bytes Free](image)

Figure 8-18  SOA bytes free
Requested bytes
The number of requested bytes gives an impression about the sizes of the objects that are created. It is just an indication, because you just see the memory request that causes a Garbage Collection. In this perspective, you can also look in the Garbage Collection log (or the CSV file produced by our GC Analyzer application) to see in which area the allocation failure occurs. Check the value of LOA, SOA, Nursery, and Tenured free bytes percent before GC. If a value is zero, that area suffers from a lack of free memory. See Figure 8-19.

Finalized objects on queue
Finalized objects are objects that were created with a finalize method by the application. Objects will be eligible for Garbage Collection after the finalizer has ran, because it is uncertain when finalizers will run (if they will run anyway).

Processing finalized objects causes overhead to the Garbage Collection and extends the “stop the world” time.

In our test, a long queue of finalized objects arose. The application has to be changed to avoid these long lists.
See Figure 8-20

![Finalized Objects in Queue](image)

**Figure 8-20  Finalized objects in queue**

**Miscellaneous error messages**

In the Garbage Collection log, you can find messages that can point you to a specific problem. Some messages are informational, while others may need your attention. Some messages we noticed during our tests:

- Insufficient free space following gc
- Compact to meet allocation
- Insufficient time being spent in gc
- Excessive time being spent in gc
- Compact to aid heap contraction
- Excessive time being spent scavenging
8.4 Just in Time (JIT) compilation

The Just In Time (JIT) is not a part of the JVM, but it works close together with the JVM. Java is an interpreted language and compilation can enhance the performance substantially.

There are not many parameters to change the behavior of the JIT, and during our test we have not changed the default values of these parameters.

For WebSphere Application Server, the JIT must be on (default on: -XJIT).

At startup of the JVM, thousands of methods have to be compiled and this will slow down the startup. Therefore, the JIT will not (by default) compile every method. It uses a call counter for every call to a method, and when this call exceeds the JIT threshold, the JIT will set the counter to zero and compile the method. From now on, every call to the method will use the compiled code. The calls to the compiled code will be counted and after reaching the threshold again, the JIT will recompile the method again with stronger optimizing rules. This will happen every time the threshold is reached, and the outcome of this is that the most used code is optimized at the highest level.

Some methods never will be compiled, even if the JIT is switched on.
Figure 8-21 shows the flow of the Just In Time compilation.
z/OS subsystems

WebSphere Application Server performance is dependent to some degree on how well the z/OS subsystems are tuned. To reach optimal performance, those z/OS subsystems should be checked. In this chapter, we provide the best practices to check those systems. The chapter is organized as follows:

- 9.1, “UNIX Systems Services” on page 218 discusses the important parameters and settings in the UNIX System Services environment.
- 9.2, “TCP/IP” on page 220 provides a few best practices on TCP/IP.
- 9.3, “Workload Manager (WLM)” on page 222 is a very important component used by WebSphere Application Server. WLM is definitely one of the most important places to look in case of performance problems.
9.1 UNIX Systems Services

WebSphere Application Server relies heavily on UNIX System Services (USS) because it uses Java, Language Environment (LE), C and C++, and TCP/IP sockets. All WebSphere Application Server configuration Data, as well as the deployed applications, are stored in an HFS. Here is a list of things you should consider in order to optimize performance in this area:

- **Mount shared file systems R/O - Read Only (R/O)**

  The SMP/E install HFS (that is, the root directory pointed to by the smpe.install.root WebSphere variable) can be mounted R/O. Having an HFS mounted R/O improves performance. However, you have to be careful with configuration HFS (that is, the root directory pointed to by was.install.root WebSphere variable). While it is possible to mount it R/O as well, and maybe this is something you want to consider for your production environment, it is not easily manageable. You will have to make sure you are pre-compiling JSPs and you are using logger for XA partner log. You will not be able to make any type of configuration changes or installation/reinstallation of the applications until you re-mount the HFS R/W. The list of possible problems having R/O HFS might be longer depending on what your applications are doing. You should test this environment first.

- **Consider using zFS, which has improved file system access. You can read more about zFS in z/OS Distributed File Service zSeries File System Implementation z/OS V1R7, SG24-6580.**

- **Determine which files are good candidates for caching with SMF 92 records. Configure the cache size in the BPXPRMxx member.**

- **Use the filecache command to cache read-only frequently accessed files. They are cached in the USS kernel. Do not use it if you have zFS. This option was deprecated in z/OS 1.7 because it does not provide any benefit over the zFS caching or might be even making it worse due to pressure on real storage. Here's an example of how to add a file to the cache:**

  ```bash
  /usr/sbin/filecache -a /WebSphere/V6R1/AppServer/properties/123.prop
  ```
  
  You can read more about this in UNIX System Services Command Reference, SA22-7802.

- **Review the parameters in the SYS1.PARMLIB(BPXPRMxx) member. Set MAXFILEPROC and MAXSOCKETS at a minimum to 5000 for low throughput, 10000 for medium throughput, and 35000 for high throughput WebSphere transaction environments. Setting high values for these parameters should not cause excessive use of resources unless the sockets or files are actually allocated.**
- MAXSOCKETS for domain AF_INET: You can monitor how many sockets are in use at any given time by issuing the `DOMVS,P` command from the MVS console. See Example 9-1, which shows that there are 417 sockets currently open (OPNSOCK) out of 25000 maximum available (MAXSOCK).

Example 9-1  Monitor socket usage

```
  D OMVS,P
  BPX0046I 12.26.56 DISPLAY OMVS 887
  OMVS    000E ACTIVE        OMVS=(6C)
  PFS CONFIGURATION INFORMATION
  PFS TYPE  DESCRIPTION          ENTRY      MAXSOCK  OPNSOCK  HIGHUSED
  NFS       REMOTE FILE SYSTEM   GFSCINIT
  CINET     SOCKETS AF_INET      BPXTCINT    25000     417     3782
  ...
```

- MAXFILEPROC: Maximum number of file descriptors that can be allocated. This includes access to HFS files and USS socket descriptors. You can monitor usage with the `DOMVS,L,PID=<decimalNumber>` command from the MVS console. PID for a given server can be obtained from server job output, for example:

  BBOJ0051I: PROCESS INFORMATION: STC28459/PFSR01A, ASID=272(0x110), PID=51316361(0x30f0689).

See Example 9-2, which shows 188 file descriptions in use out of 65535 available.

Example 9-2  Monitor file descriptor usage

```
  D OMVS,L,PID=51316361
  BPX0051I 12.59.28 DISPLAY OMVS 277
  OMVS    000E ACTIVE        OMVS=(6C)
  USER     JOBNAME  ASID        PID       PPID
  ASCR1    PFSR01A  0110   51316361          1
  LATCHWAITPID=         0 CMD=BBOCTL
  PROCESS LIMITS:        LIMMSG=SYSTEM
  CURRENT  HIGHWATER    PROCESS
  USAGE      USAGE      LIMIT
  MAXFILEPROC           188        291      65535
  ...
```
9.2 TCP/IP

WebSphere Application Server uses the TCP/IP sockets communication mechanism extensively. Some configuration changes can improve performance notably. Here are some things to look at:

- Ensure you have defined enough sockets. In BPXPRMxx, review the following two variables. See 9.1, “UNIX Systems Services” on page 218 for more details about these variables and recommended settings.
  - MAXFILEPROC
  - MAXSOCKETS

- Review the following definitions in TCPIP.PROFILE:
  - Consider specifying NODELAYACKS for ports. This may improve throughput for more trivial transactions, but it does add more overhead for more complex transactions. Using this option will cause the acknowledgments (ACKs) to be sent back immediately to the client, rather than waiting for more data to accumulate. An example of this setting is:
    PORT 8080 TCP NODELAYACKS
  - Increase TCPSENDBFRSIZE and TCPRCVBUFRSIZE. These variables define the size of the send and receive buffers. The default setting is 16 KB, which should be increased to at least 64 KB (256 KB is the maximum and could also be reasonable in certain cases). An example of this setting is:
    TCPCONFIG TCPSENDBFRSIZE 65535
    TCPRCVBUFRSIZE 65535
  - If, after increasing the socket definitions as mentioned above, you still have problems with socket usage, reduce the FINWAIT2TIME. When a socket is closed abnormally (for example, no longer needed), it is placed into a state called finwait2. It will remain in this state for a default of 600 seconds (plus an additional 75 seconds) before the connections is released and made available. An example of this setting is:
    TCPCONFIG FINWAIT2TIME 60

You can see if you have a lot of connections in findwait2 state by issuing the following command from tso session or omvs:

netstat -s
Increase the default listen backlog, which is used to specify the maximum length for the queue of pending connections using the given protocol. The default is set to 10 requests, but can be increased to allow for cases where there are spikes in new connections. The following listen backlog variables can be added/modified in the WebSphere Admin Console under Servers → Application servers → ServerName → Server infrastructure → Administration → Custom Properties, where you add the following variables:

- protocol_http_backlog=100
- protocol_https_backlog=100
- protocol_iiop_backlog=100
- protocol_ssl_backlog=100

The values for above variables are limited by the SOMAXCONN setting in TCPIP.PROFILE.

Consider using HiperSockets™ for IP traffic in a sysplex (server-to-server communication). You can read more about it in the document HiperSockets: A Dramatic Increase in Networking Bandwidth at the following site:


You can also refer to HiperSockets Implementation Guide, SG24-6816.

DNS configuration

- Ensure that lookups for frequently-used servers and clients are being cached. Sometimes caching is related to the TTL (Time To Live) value. However, setting this value too high might not always be good, as it will take longer for the system to be aware of network changes (for example, if Daemon goes down).

- Lower the RESOLVERTIMEOUT value in TCPIP.DATA. The default is 30 seconds. If there are some problems with DNS definitions (either there are old entries or there are problems with the particular DNS server and it is not responding), the name or address resolution can be delayed for 30 seconds for each bad entry. An example of the setting is:

  RESOLVERTIMEOUT 10
9.3 Workload Manager (WLM)

Workload Manager (WLM) is used by WebSphere Application Server for z/OS to manage workload distribution between servants and to start and terminate servants in response to workload activity.

First, the request comes into the controller address space. The controller creates an enclave and associates the transaction to this classified enclave. The transaction is queued to the WLM queue and waits to be served by an available WebSphere worker thread in a servant address space. Tuning WLM is important to make sure these operations go smoothly and CPU is fully utilized.

9.3.1 WLM classification

How you configure WLM classification will affect overall application throughput. WebSphere Application Servers should be set to a relatively higher priority. Here are some suggestions on how to classify the workload.

Controllers and daemons

Controllers include Deployment Manager Controller, Node Agent, and Application Server Controller address spaces. Controllers and daemons should be classified as SYSSTC or high velocity under STC classification rules. The main function of the controller is routing work to servants with minimal other processing, so they need higher priority. To verify and modify your settings for STC, go to the WLM screen, as shown in Figure 9-1:

- On the main WLM screen, choose option 6 - Classification Rules and choose STC type.

<table>
<thead>
<tr>
<th>Subsystem Type Selection List for Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action Codes: 1=Create, 2=Copy, 3=Modify, 4=Browse, 5=Print, 6=Delete, /=Menu Bar</td>
</tr>
<tr>
<td>------Class------</td>
</tr>
<tr>
<td>Action</td>
</tr>
<tr>
<td>__</td>
</tr>
<tr>
<td>__</td>
</tr>
</tbody>
</table>

Figure 9-1  WLM screen - option 6 Classification Rules
Create new rules, as shown in Figure 9-2. In this example, Controller PFSR01 will run in SYSTC service class, and servants will use VEL80 service class.

![Figure 9-2  STC Classification Rules - Modify view](image)

**Servants**

Servants should also be classified high (possibly SYSTC), but they could be lower than controllers under STC classification rules (Figure 9-2). The service class defined here is used for tasks that run in the servant region under control of the step task and not as part of the enclave.

**Note:** Java Garbage Collection as well as JavaServer Pages (JSP) compiles run under STC classification.

The work in the servants that runs under an enclave is classified using CB subsystem type (see Figure 9-1 on page 222).

The classification can be based on the following WLM criteria:

<table>
<thead>
<tr>
<th>CN</th>
<th>Collection Name: Either “server_generic_short_name” or “Cluster Transition Name” (equivalent to your WLM application environment name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SI</td>
<td>Server Instance Name: “server_specific_short_name”</td>
</tr>
<tr>
<td>UI</td>
<td>User ID: The user ID assigned to the transaction</td>
</tr>
<tr>
<td>TC</td>
<td>Transaction Class: Defined in the transaction class mapping file for the server</td>
</tr>
</tbody>
</table>
An example of CB classification is shown in Figure 9-3. In this case, we also used the Transaction class (TC) for more granular classification based on type of incoming work request: HTTP, IIOP, or message-driven bean (MDB). These should be defined in a class mapping file - classification.xml (pointed to by the wlm_classification_file variable). Each transaction_class defined in the mapping file has to be assigned to a service class.

**Note:** Each servant (SR) address space can handle work only for one particular service class.

```
Modify Rules for the Subsystem Type
Command ===>

Subsystem Type : CB    Fold qualifier names?  Y
Description . . WebSphere App Server
Action codes: A=After   C=Copy   M=Move   I=Insert rule
             B=Before   D=Delete row  R=Repeat   IS=Insert Sub-rule

--------Qualifier-------- -------Class--------
Action Type Name Start Service     Report
DEFAULTS: WASDF       OTHER
___  1  CN     PFSR03 ___ WASSR3AF    ________
___  2  TC     CLASS1 ___ WASHI ________
___  2  TC     CLASS2 ___ WASLO ________
___  2  TC     CLASS3 ___ WASDF ________
```

*Figure 9-3  CB Classification Rules - Modify View*

When defining transaction classes, make sure you are also configuring the Minimum Number of Instances and Maximum Number of Instances properties. Once a new servant accepts a first request that is classified under one service class, this servant is bound to this service class and can accept future requests only for this one service class. That is why you may want to define “Minimum Number of Instances” as this:

Minimum Number of Instances = number of expected service classes

so each request that comes in has a servant available and a new one does not need to be started. If you need to define the wlm_maximumSRCount variable, you need to make sure you set it high enough so that there will be enough servants to handle work for each service class.

For instance, let us consider a situation where you have two service classes: one that is “busy” (a large number of requests are running under this class) and one
that is “not busy” (used infrequently), and you also have a maximum of five servant regions. If there is a large workload for the “busy” service class, all five servants might be bound to this one ‘busy’ service class. If a requests comes in for the second “non-busy” service class, WLM will not be able to dispatch this request because none of the five servants are bound to the service class this request needs to run under and WLM is not allowed to start any new service classes.

You can read more about the SRCount variables in 7.3.3, “Controlling the number of servants” on page 172.

Once transaction classes are configured and in effect, you can monitor how your requests are being classified with a dynamic display command from the MVS console. The requests are broken down by HTTP and IIOP type. This command is not designed to show MDB work classification:

```
F <ServerName>,DISPLAY,WORK,CLINFO
```

```
BB000281I CLASSIFICATION COUNTERS FOR IIOP WORK
BB000282I CHECKED 14, MATCHED 14, USED 0, COST 0, DESC: IIOP Default
BB000282I CHECKED 14, MATCHED 3, USED 0, COST 0, DESC: sample
BB000282I CHECKED 3, MATCHED 1, USED 1, COST 3, DESC: ala
BB000282I CHECKED 2, MATCHED 1, USED 1, COST 4, DESC: alb
BB000282I CHECKED 1, MATCHED 1, USED 1, COST 5, DESC: alc
BB000282I CHECKED 11, MATCHED 11, USED 0, COST 0, DESC: other
BB000282I CHECKED 11, MATCHED 1, USED 1, COST 4, DESC: a
BB000282I CHECKED 10, MATCHED 1, USED 1, COST 5, DESC: b
BB000282I CHECKED 9, MATCHED 1, USED 1, COST 6, DESC: c
BB000282I CHECKED 8, MATCHED 2, USED 2, COST 7, DESC: d
BB000282I CHECKED 6, MATCHED 1, USED 1, COST 8, DESC: e
BB000282I CHECKED 5, MATCHED 4, USED 4, COST 9, DESC: f
BB000282I CHECKED 1, MATCHED 1, USED 1, COST 10, DESC: g
BB000283I FOR IIOP WORK: TOTAL CLASSIFIED 14, WEIGHTED TOTAL COST 95
BB000281I CLASSIFICATION COUNTERS FOR HTTP WORK
BB000281I CHECKED 0, MATCHED 0, USED 0, COST 0, DESC: HTTP Default
BB000282I CHECKED 0, MATCHED 0, USED 0, COST 0, DESC: n
BB000282I CHECKED 0, MATCHED 0, USED 0, COST 0, DESC: o
BB000282I CHECKED 0, MATCHED 0, USED 0, COST 0, DESC: q
BB000283I FOR HTTP WORK: TOTAL CLASSIFIED 0, WEIGHTED
BB000188I END OF OUTPUT FOR COMMAND DISPLAY,WORK,CLINFO
```
Define the service class for the CB subsystem with a response goal percentile that is reasonable in your environment. For example, a goal that 80% of the work will complete in .25 seconds is a typical goal. Velocity goals for application work are not meaningful and should be avoided. See Figure 9-4 for a sample definition of a service Class.

**Note:** Do not set a goal that is too aggressive, as WLM might decide to ignore your request in favor of the rest of the system.

If the goal you set is high and not achievable, WLM will stop trying to reach this goal and will instead give more attention to the rest of the system, and the application servers will suffer.

```
Modify a Service Class
Service Class Name . . . . . : WASSR1AE
Description . . . . . . . . . 6.0 PESR0* servers
Workload Name . . . . . . . WASPERFW (name or ?)
Base Resource Group . . . . (name or ?)
Cpu Critical . . . . . . . . . NO (YES or NO)

Specify BASE GOAL information. Action Codes: I=Insert new period, E=Edit period, D=Delete period.

---Period--- ---------------------Goal-------------------
Action  #  Duration   Imp.  Description
__  1              1    90% complete within 00:00:00.750
```

*Figure 9-4  Example of Service Class definition that is used under CB subsystem*

For more details on WLM configuration, see *z/OS MVS Planning: Workload Management* SA22-7602. *System Programmer’s Guide to: Workload Manager, SG24-6472* has more WebSphere Application Server specific information.
9.3.2 Temporal affinity workload balancing

A problem with workload balancing can occur when multiple servants are running and they belong to the same service class. WLM is responsible for dispatching requests to the servants in a way that the goal that was set is met. One servant may get more work (within its capacity) as WLM will also try to bring down any additional servants if they are not necessary for handling the given workload in order to consume less resources. This is regardless of to what WebSphere wlm_minimumSRCount is set. This can cause a problem when one servant region receives the majority of the requests that require temporal affinity (the “matching” request has to be handled by the same servant when it comes back). Work might be waiting on the WLM queue even though there are servants available. Requests that generated affinity work are typically servlet requests with http session objects or stateful session EJBs.
An example of this condition is shown in the top half of the Figure 9-5. The example assumes there are three servants (SRs) active and each servant has three worker threads. There are four new requests that need to be picked up by the servants. Two of them are new requests that do not have session objects established (though they might or might not establish a new session objects once dispatched in SR), so they will most likely be dispatched to the next least busy SR2. However, the other two new requests (R1 and R6) need to go back to the servant where the temporal affinity was already created - SR1. Unfortunately, SR1 has all three threads busy and therefore both requests are put on the special WLM queue for this SR1 address space, even though there are still available threads in SR3.

Figure 9-5  WLM behavior with and without WLMStatefulSession enabled.
This situation could be avoided by setting a WebSphere variable that changes WLM behavior. It enables workload balancing based on the number of temporal affinities associated with each servant. A new request goes to the servant with the smallest number of affinities associated with it. If there are no temporal affinities, WLM will try to balance the workload based on the number of active requests in the servant. Therefore, this variable might also help if you see problems with workload balancing even if your application does not use session objects. However, remember that the original WLM behavior changes when this variable is set and WLM will be less likely to bring down unnecessary servant’s address.

**Note:** The WLMStatefulSession variable might help when you see problems with workload distribution between servants, even in a case where there is no temporal affinities.

In order to enable the affinity workload distribution, set the following property:

- In the WebSphere Admin Console, select **Servers** → **Application Servers** → `<yourServer>` → **Server Infrastructure** → **Administration** → **Administration Services** → **Customer Properties**.

- Change the following property to true:
  
  Name: WLMStatefulSession
  
  Value: true

- Select **Apply** → **Save** and then **Synchronize**.

- Restart the application server.

**Note:** In WebSphere Application Server Version 5.1, the WLMStatefulSession variable equivalent is `wlm_stateful_session_placement_on` and was defined under **Environment** → **Manage WebSphere variables**.

To make sure you make full use of this functionality, you should ensure there is always a number of servants available (otherwise, there is no reason to distribute work). You can do this with the Number of Instances setting described in 7.3.3, “Controlling the number of servants” on page 172.

The example of WLM behavior with WLMStatefulSession enabled is shown in the bottom half of the Figure 9-5 on page 228. In this case, the previous requests that created temporal affinity were distributed between different servants. All four new requests can be dispatched to SRs and there is no WLM queue wait. Notice that the two new requests that are not associated with temporal affinity are likely going to be dispatched to two different SRs in case they do create a temporal affinity.
However, do not expect even distribution between all servants. This is not a round-robin distribution. WLM considers different factors when dispatching requests and the above example is just a simple scenario. This solution has also some problems. If each servant has already a number of temporal affinities defined and one of them goes down (timeout or other reason), the newly created servant will now have no temporal affinities defined. This new servant will likely be more busy than others for a period of time until number of affinities evens out between servants.

9.3.3 RMF Workload Activity Report

Resource Measurement Facility (RMF) Workload Activity Report can give you very good information about which area on your system might be causing problems. It is helpful in situations when your requests are not handled quickly enough, the CPU is not fully utilized, or you think WebSphere Application Server is not responding. RMF Reports show you a z/OS point of view on the system as opposed to the application view that many other tools reviewed in this book do.
RMF Workload Activity Report is based on SMF records 70-79, which are usually enabled all the time. A sample report is shown in Figure 9-6.
You can find detailed information about how to read Workload Activity Report in *z/OS Resource Measurement Facility Report Analysis*, SC33-7991. Here are some areas you might want to take a look at. The number in the list corresponds to the number displayed in Figure 9-6 on page 231:

1. Identifies the interval that was used to generate the report. In this case, the report represents workload during a five minute period.

2. The reports are grouped by either service class or report class. This example shows a report for WASSR1AE service class. It is a good idea to define different application servers to different service classes (that could have the same response goal). In this case, WASSR1AE is only used by one application server. You could also define a different report class instead, but the report class view in Workload Activity Report includes less information in comparison to the service class view.

3. Define how critical the work in this service class is in comparison to other things going on in the system.
   a. CPU or Storage Critical flag.
      When you define a service class, use CPU Critical only when you know the work on the application server is very CPU intensive and you suspect other applications are taking away too much CPU time from the WebSphere Application Server.
   b. An importance of 1 describes the highest priority class for most important work (5 being the lowest).

4. The goal that was defined for this service class was 90% of transactions should complete within 0.75 seconds or less.

5. The actual percentage of transactions that did finish within the specified goal. In this case, we did not reach the goal: 76.1% instead of a defined goal of 90%.

6. If the goal is not met, you will probably see some delays in this area (note that if there are no delays, this section is blank). In this case, 1% of transactions was waiting for CPU and 51.6% was waiting on the WLM queue. Based on this result, it looks like WLM for some reason did not or was not able to dispatch the requests quickly enough. With only 1% CPU of delay, it does not seems like the reason was that CPU was overloaded.

7. APPL% CP
   The percentage of CPU time used by transactions running on a single CPU. So, if you have three CPUs and each one was running at 100% during this reporting period, this number would equal to 3 * 100% = 300%. The system this report was generated on had three CPUs. Only 31.4% of the CPUs out of 300% available was utilized in this period. This further confirms the missed goal was not caused by the CPUs being contained!
If, on the other hand, CPU delay (QMPL - see number 6 above) was high, and APPL% CP was still low, this could indicate that the WLM goal is set to too aggressive and WLM gave up trying to reach it.

8. This section gives you a summary of what happened during this reporting interval.
   - AVG = MPL = AVG ENC: Average number of transactions during the interval (transaction active time divided by reporting period interval). In WebSphere, each transaction will have an enclave, so these numbers should be equal.
   - ENDED: Number of transactions that ended in this reporting interval.

9. ACTUAL = EXECUTION + QUEUED
   - EXECUTION: The average execution time of ended transactions.
   - QUEUED: Average time the job was delayed.

10. If there is any paging going on, it will be listed in this section. In this case, there was no paging.

11. This section gives you a better view of how many transactions ran within the goal and how many ran longer. In this example, out of total of 2178 requests, 1657 ran within the goal.

The problem represented in the RMF report in Figure 9-6 on page 231 was caused by a small number of worker threads being available to handle the incoming workload. There was enough CPU available to handle more work. The major delay was caused by requests waiting on the WLM queue. Once the number of worker threads was increased (see 7.3.2, “Workload Profile” on page 171), the RMF report showed few delays and the goal was being met.

### 9.4 Resource Recovery Services (RRS)

WebSphere Application Server uses Resource Recovery Services (RRS) for global transaction processing. RRS writes its data to a logstream. Major improvements in RRS processing can be gained by ensuring the logging process is optimized.

- Use Coupling Facility (CF) for logstream instead of DASD, especially for main and delayed logs that contain live or active data.

- If you need to use DASD-only logging, use the highest performing DASD possible. Allocate logs with large CI sizes.
If using CF, ensure offloading is not occurring. Proper sizing of the RRS logs is important. Too small and you get reduced throughput, since logger is off-loading the logs too frequently. Too large and you could overflow your Coupling Facility. Table 9-1 shows some recommended settings for logstream sizes. You can further tune the sizes using SMF record type 88.

<table>
<thead>
<tr>
<th>Logstream</th>
<th>Initial size</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>RM DATA</td>
<td>1 MB</td>
<td>1 MB</td>
</tr>
<tr>
<td>MAIN UR</td>
<td>5 MB</td>
<td>50 MB</td>
</tr>
<tr>
<td>DELAYED UR</td>
<td>5 MB</td>
<td>50 MB</td>
</tr>
<tr>
<td>RESTART</td>
<td>1 MB</td>
<td>5 MB</td>
</tr>
<tr>
<td>ARCHIVE</td>
<td>5 MB</td>
<td>50 MB</td>
</tr>
</tbody>
</table>

XA Partner Log and Compensation Service can be configured to use a logstream. From a performance point of view, it is better to use HFS for XA Partner Log. There is also no point of setting up a logstream for XA resources if you are not using any (for example, Universal JDBC Provider with XA Datasources). See 7.3, “WebSphere Application Server” on page 170 for more information about configuring log compression interval.

Consider eliminating ARCHIVE log if not needed. The archive log contains the results of completed transactions. Normally, the archive log is not needed unless you are developing and tuning the application.

9.5 Language Environment (LE)

Language Environment (LE) provides a common runtime environment for high-level programming languages like C, C++, or COBOL. The following are some areas to look into, especially for production systems:

- Load LE and C++ reentrant runtimes into LPA. SYS1.PARMLIB(LPALSTxx) should have the following defined:
  CEE.SCEELPA
  CBC.SCLBDLL

- Non-reentrant LE routines should be in the link list.
  SYS1.PARMLIB(LNKLSTxx) should include the following:
  CEE.SCEERUN
Ensure that the LE data sets, SCEERUN and SCEERUN2, are authorized to enable xplink.

Processes that run the client ORB, since they start the JVM, must run with XPLINK(ON). For best performance, compile applications that use JNI services with xplink enabled. Compiling applications with xplink enabled improves performance in z/OS V1R2. As you move from z/OS V1R2 to z/OS V1R6, you should experience additional performance improvements when all of the LE services calls are xplink enabled.

Use the following options only for tuning purposes. Once you are done with tuning, remove these options, as they degrade performance.

- **RPTOPTS(ON)**: A report of the runtime options was in effect while the application was running.
- **RPTSTG(ON)**: A report of the storage the application used to help in setting the heap size, heappools, and so on.
- **HEAPCHK(ON)**: Used in combination with other suboptions in order to enable heap checking or tracing.

**Note:** Some LE parameters are already set by WebSphere Application Server internally to maximize performance. These options can be overridden in the JCL of the started task, but you may need to consult IBM to ensure the internal settings are not compromised.

Turn LE Heappools on in the client running on z/OS (they are already on for the WebSphere Application Server by default). Use the following in the shell script:

```
SET LEPARM='HEAPP(ON)'
```

### 9.6 Other considerations

A few additional considerations are discussed in the following sections.

#### 9.6.1 GRS

WebSphere Application Server uses *Global Resource Serialization (GRS)* to communicate between servers and sysplex. If you are using global transactions, WebSphere will use enqueues (ENQ) and dequeues (DEQ) to serialize global transactions. Set the following GRS options based on your configuration:

- If you are in a sysplex, use GRS=STAR.
- If you are *not* in a sysplex, use GRS=NONE.
9.6.2 Storage

WebSphere Application Server on z/OS requires relatively more storage than most other servers running on z/OS. One reason is a high usage of Java within WebSphere. You need to ensure that virtual storage is not limited and there is enough real storage available.

Virtual storage constraints are less of a concern if 64-bit mode is used (available with WebSphere Application Server V6.1). However, the more storage you use, the more expensive it is to manage it.

- Review the REGION setting on the JCL procedure. By default, REGION is set to 0M, which means "take as much as needed". However, in some customer environments, there may be a default REGION limit set and 0M will not overwrite this. You may need to set a specific number instead.

- You will need more REGION if your procs are STEPLIBing to WebSphere load modules. We recommend moving WebSphere modules to LPA (common storage).

- If you need a large JVM heap size, consider setting REGION to a specific number instead of 0M. The z/OS operating system allocates user storage from the bottom of the address space, which is where the JVM heap is allocated, and system storage from the top. System abends can occur when the system tries to obtain virtual storage and none is available. A non-zero REGION parameter setting prevents this from occurring by preserving storage at the top of the address space for system use.

- Review the system exits that limits virtual storage usage (for example, IEFUSI, JES, or TSO segment defaults). Make sure they are either not set or set high enough. The amount of storage greatly depends on JVM Heap Size. The storage usage will also increase if you are using STEPLIB or LinkList for WebSphere modules.

- One way to see if your storage is constrained is to look at RMF reports that shows paging rates. See 9.3.3, "RMF Workload Activity Report" on page 230. You can also view current paging rates as well as REAL storage usage in the SDSF.DA (Active users) output:

```
    JOBNAME   Real Paging   SIO   CPU%   ASID   ASIDX
    BBODMNC   3026   0.00   0.00   0.00   27   001B
    BBOS001S  96T    0.00   0.00   0.89   55   0037
    BBOS001   58T    0.00   0.00   0.89   58   003A
```
Connectors

Most WebSphere applications use back-end transaction systems, databases systems, or WebSphere MQ. Performance problems experienced in WebSphere can be the result of a bad setup of a connector to CICS or IMS, JDBC to a relational database, or connectivity between WebSphere Application Server and WebSphere MQ.

Attention: Performance problems experienced in WebSphere Application Server can also be the result of the behavior of the back-end system itself, but this is beyond the scope of this book.

The chapter is organized as follows:

- 10.1, “WebSphere back-end connectivity” on page 239 is an introductory section to connectivity in general.
- 10.2, “Java Database Connectivity (JDBC)” on page 243 deals with the most important widely used area of connectivity: connectivity between WebSphere and the relational database using JDBC.
- 10.3, “Java Message Service (JMS) and WebSphere MQ” on page 250 addresses some best practices related to the usage of WebSphere MQ as the messaging provider.
- 10.4, “CICS Transaction Gateway” on page 257 and 10.5, “IMS Connect” on page 261 discuss best practices for the J2C connectors to CICS and IMS respectively.
10.6, “Using the Performance Monitoring Infrastructure (PMI)” on page 263 is a chapter on using PMI for J2C connectors and JDBC.
10.1 WebSphere back-end connectivity

In WebSphere Application Server on z/OS, **J2C connectors** are the means for connecting to Enterprise Information Systems (EIS). **JavaDatabase Connectivity (JDBC)** is the combination of APIs and infrastructure to connect to an RDBMS. **Java Message Service (JMS)** is the API to use for messaging systems, such as WebSphere MQ.

We will not discuss the architectural merits of the different types of connections here, for example, local or remote connections, as these are covered in detail in *WebSphere for z/OS Connectivity Architectural Choices*, SG24-6365.

There are two separate **Connection Managers** (JDBC and J2C) for WebSphere on z/OS, and we will examine the properties of both these different types of connectivity and highlight how they can be tuned to improve performance.

Each time an application attempts to access a back-end system (such as a database or EIS), it requires resources to create, maintain, and release a connection to that system. To mitigate the strain this process can place on overall application resources, WebSphere Application Server enables administrators to establish a pool of back-end connections that applications can share on an application server. **Connection pooling** spreads the connection overhead across several user requests, thereby conserving application resources for future requests.

Connection pooling can improve the response time of any application that requires connections, especially Web-based applications. When a user makes a request over the Web to a resource, the resource accesses a datasource or some other form of **connection factory**. Because users connect and disconnect frequently with applications on the Internet, the application requests for data access can surge to considerable volume. Consequently, the total resource access overhead quickly becomes high for Web-based applications, and performance deteriorates. When connection pooling capabilities are used, however, Web applications can realize performance improvements of up to 20 times the normal results. With connection pooling, most user requests do not incur the overhead of creating a new connection, because the resource can locate and use an existing connection from the pool of connections. When the request is satisfied and the response is returned to the user, the resource returns the connection to the connection pool for reuse. The overhead of a disconnection is avoided. Each user request incurs a fraction of the cost for connecting or disconnecting. After the initial resources are used to produce the connections in the pool, additional overhead is insignificant, because the existing connections are reused.
A separate instance of a given configured connection pool is created on each application server that uses that datasource or connection factory. For example, if you run a three server cluster in which all of the servers use myDataSource, and myDataSource has a maximum connections setting of 10, then you can generate up to 30 connections (three servers times 10 connections). Be sure to consider this fact when determining how many connections to your back-end resource you can support.

Back-end connectivity falls into three categories as far as the WebSphere Administrative Console is concerned:

- **Java DataBase Connectivity**
  For access to databases via JDBC or SQLJ. As the name suggests, this is a JDBC connector.

- **Java Message Service**
  For message-based access to EIS systems. This is normally an asynchronous mode of operation, and our focus in this book is on WebSphere MQ as the JMS Provider. This is now also a J2C connector.

- **Resource Adapters**
  For synchronous access to EIS systems, in particular CICS and IMS. These are both examples of J2C connectors.

### 10.1.1 Common Connection pool properties

All of these connectors use a common set of connection pool properties and these are discussed below.

**Connection Timeout**
This value specifies the number of seconds a request for a connection waits, when there are no connections available in the free pool and no new connections can be created, usually because the maximum value of connections in the particular connection pool has been reached.

For example, if Connection Timeout is set to 180, and the maximum number of connections are all in use, the pool manager waits for 180 seconds for a physical connection to become available. If a physical connection is not available within this time, the pool manager initiates a ConnectionWaitTimeout exception.

If Connection Timeout is set to 0, the pool manager waits as long as necessary until a connection becomes available. This happens when the application completes a transaction and returns a connection to the pool, or when the number of connections falls below the value of Maximum Connections, allowing a new physical connection to be created.
If Maximum Connections is set to 0, which enables an infinite number of physical connections, then the Connection Timeout value is ignored.

The default value is 180.

**Maximum Connections**
This value specifies the maximum number of physical connections that you can create in a pool. These are the physical connections to a back-end resource.

Once this number is reached, no new physical connections are created and the requester waits until a physical connection that is currently in use returns to the pool, or a ConnectionWaitTimeout exception is issued.

For example, if the Maximum Connections value is set to 10, and there are ten physical connections in use, the pool manager waits for the amount of time specified in Connection Timeout for a physical connection to become free.

If Maximum Connections is set to 0, the connection pool is allowed to grow infinitely. This also has the side effect of causing the Connection Timeout value to be ignored.

If multiple stand-alone application servers use the same datasource, there is one pool for each application server. If clones are used, one data pool exists for each clone. Knowing the number of data pools is important when configuring the database Maximum Connections.

The default value is 10.

**Minimum Connections**
This value specifies the minimum number of physical connections to maintain in the pool.

If the size of the connection pool is at or below the minimum connection pool size, then the Unused Timeout setting is ignored and no physical connections are discarded.

If Aged Timeout is set to a non-zero value, then connections with an expired age are discarded, regardless of the minimum pool size setting.

New physical connections will not be created solely to ensure that the minimum connection pool size is maintained. This also means that the pool is not automatically filled to its minimum level when an instance of the Application Server is started.

The default value is 1.
**Reap Time**
This value specifies the interval, in seconds, between runs of the pool maintenance thread. For example, if Reap Time is set to 180, the pool maintenance thread runs every 180 seconds.

If the pool maintenance thread is enabled, that is, the Reap Time value is greater than zero, then this value should be less than the values of Unused Timeout and Aged Timeout. When the pool maintenance thread runs, it discards any connections remaining unused for longer than the time value specified in Unused Timeout, until it reaches the number of connections specified in Minimum Connections. The pool maintenance thread also discards any connections that remain active longer than the time value specified in Aged Timeout.

The Reap Time interval affects the accuracy of the Unused Timeout and Aged Timeout settings. The smaller the Reap Time interval, the greater the accuracy of these other settings.

The Reap Time interval also affects performance. Smaller intervals mean that the pool maintenance thread runs more often and degrades performance.

To disable the pool maintenance thread, set Reap Time to 0, in which case Unused Timeout and Aged Timeout are ignored.

The default value is 180.

**Unused Timeout**
This value specifies the interval, in seconds, after which an unused or idle connection is discarded.

The Unused Timeout value should be set to a value higher than the Reap Time value for optimal performance. Unused physical connections are only discarded if the current number of connections exceeds the Minimum Connections setting.

For example, if the unused timeout value is set to 1800, and the Reap Time value is not 0, any physical connection that remains unused for thirty minutes is discarded.

The default value is 1800.

**Aged Timeout**
This value specifies the interval in seconds before a physical connection is discarded.

If Aged Timeout is set to 0, active physical connections remain in the pool indefinitely.
If the Aged Timeout value is non-zero, then it should be set to a value higher than the Reap Time value for optimal performance.

For example, if the Aged Timeout value is set to 1200, and the Reap Time value is not 0, any physical connection that remains in existence for 1200 seconds (20 minutes) is discarded from the pool. The only exception is if the connection is involved in a transaction when the aged timeout is reached. If it is, the connection is closed immediately after the transaction completes.

The default value is 0.

**Purge Policy**

Specifies how to purge connections when a stale connection or fatal connection error is detected.

Valid values are as follows:

- **EntirePool**
  All connections in the pool are marked stale. Any connection not in use is immediately closed. A connection in use is closed and issues a stale connection Exception during the next operation on that connection.

  When using this purge policy, there is a slight possibility that some connections in the pool are closed unnecessarily when they are not stale. However, this is a rare occurrence.

  In most cases, a purge policy of EntirePool is the best choice.

- **FailingConnectionOnly**
  Only the connection that caused the stale connection exception is closed. Although this setting eliminates the possibility that valid connections are closed unnecessarily, it makes recovery from an application perspective more complicated.

  The default value is EntirePool.

### 10.2 Java Database Connectivity (JDBC)

In addition to the common Connection Pool properties, there are some JDBC specific properties available for tuning via the WebSphere Administrative Console.

The tuning of these properties of data sources and connection pools optimize the performance of transactions between the WebSphere application and the datastore.
For a more detailed description of JDBC and the way it operates in DB2 for z/OS, please refer to *DB2 for z/OS and WebSphere: The Perfect Couple*, SG24-6319.

### 10.2.1 Data source tuning

The following sections discuss tuning considerations at the data source level.

**Statement cache size**

To view the WebSphere Administrative console page where you configure this property, select Resources → JDBC → JDBC Providers → <JDBC_provider_name> → Data sources → <data_source_name> → WebSphere Application Server data source properties.

This page is shown in Figure 10-1.
This property specifies the number of statements that can be cached for a given data source per connection. WebSphere Application Server caches a statement after the user closes it.

The WebSphere Application Server data source optimizes the processing of prepared statements and callable statements by caching those statements that are not being used in an active connection. Both statement types help maximize the performance of transactions between your application and datastore.

- A **prepared statement** is a precompiled SQL statement that is stored in a PreparedStatement object. WebSphere Application Server uses this object to run the SQL statement multiple times, as required by your application runtime, with values that are determined by the runtime.

- A **callable statement** is an SQL statement that contains a call to a stored procedure, which is a series of precompiled statements that perform a task and return a result. The statement is stored in the CallableStatement object. WebSphere Application Server uses this object to run a stored procedure multiple times, as required by your application runtime, with values that are determined by the runtime.

If the statement cache is not large enough, useful entries are discarded to make room for new entries. To determine the largest value for your cache size to avoid any cache discards, add the number of uniquely prepared statements and callable statements (as determined by the sql string, concurrency, and the scroll type) for each application that uses this data source on a particular server. This value is the maximum number of possible statements that can be cached on a given connection over the life of the server. Setting the cache size to this value means you never have cache discards. In general, the more statements your application has, the larger the cache should be.

**Note:** In general terms, the more statements your application has, the larger the cache should be. Be aware, however, that specifying a larger statement cache size than needed wastes application memory and does not improve performance.

You can determine the value for your cache size by adding the number of uniquely prepared statements and callable statements (as determined by the SQL string, concurrency, and the scroll type) for each application that uses this data source on a particular server. This value is the maximum number of possible prepared statements that are cached on a given connection over the life of the server.

Default: For most databases, the default is 10. Zero means there is no statement cache.
Enabling DB2 dynamic statement cache

The WebSphere dynamic statement cache function works together with DB2 dynamic statement caching. When the prepared statements are cached in the EDM statement cache, re-calculation of the access path can be avoided if the statement in the DB2 dynamic statement cache can be reused by a subsequent application. This saving is in addition to saving the JDBC precompiled SQL cost that WebSphere statement caching offers.

The following DB2 system parameters (in DSN6SPRM) should be reviewed:

- `CACHEDYN=YES` means the DB2 global dynamic statement cache is enabled.
- `MAXKEEPD` specifies the maximum number of SQL statements kept in the local statement cache. This parameter only applies to applications that also use the `KEEPDYNAMIC(YES)` bind option.
- `EDMSTMTC` specifies the size of the global dynamic statement cache. The global dynamic statement cache resides above the 2 GB bar and is always in a separate storage area from the EDM pool.

keepDynamic custom property

To view the WebSphere Administrative Console page where you configure this property, select Resources → JDBC → JDBC Providers → `<JDBC_provider_name>` → Data sources → `<data_source_name>` → Custom properties.

Recommendation: This value should be set lower than the number of threads allocated by the WebSphere Application Server Workload profile (see 7.3.2, “Workload Profile” on page 171).
The keepDynamic property determines whether DB2 keeps dynamic SQL statements after commit points. This is applicable only when the target server is DB2 for z/OS. The property default depends on the server. Specifying “1” means YES and DB2 will keep dynamic SQL statements after commit points. Specifying “2” means NO and DB2 will not keep dynamic SQL statements after commit points. If this property is not set, defaults are assumed. If the JDBC package set is custom bound without these defaults, this property must be set. The driver has no way to know how custom JDBC packages were bound under current DRDA® architecture. This property is not a directive to server, but rather a driver directive.
Care should be taken when using this property or specifying KEEPDYNAMIC(YES) as a DB2 BIND option. They should not be used globally as a means of improving overall performance, as it will overuse DBM1 virtual storage below the 2 GB bar and affect DB2 DDF connection pooling, as the connections will not go inactive.

KEEPDYNAMIC(YES) should only be used for high volume, simple, performance sensitive transactions and not as a general purpose default setting.

10.2.2 WebSphere connection pool and DB2 threads tuning

To view the WebSphere Administrative Console page where you configure the connection pool properties, select Resources → JDBC Providers → <JDBC_provider_name> → Data sources → <data_source_name> → Connection pool properties.

This page is shown in Figure 10-3.

![Figure 10-3 Data source connection pool properties](image-url)
The WebSphere connection pool settings are directly related to the number of connections that have to be configured in DB2 to support this setup. Knowing the total number of connection pools is also important when configuring the database’s maximum connections.

The following settings for DB2 threads set the limits that govern the number of WebSphere connections:

- **MAX USERS (CTHREAD parameter)** specifies the maximum number of allied threads (threads started at the local subsystem) that can be allocated concurrently. These threads are for local requests such as Type 2 JDBC users. The default value for this is 200.

- **MAX REMOTE ACTIVE (MAXDBAT parameter)** specifies the maximum number of database access threads (DBATs) that can be active concurrently. These threads are used for connections entering DB2 through, among other types of connection, remote requests via the DB2 Universal Driver using Type 4 connectivity. The default value for this is 200.

- **MAX REMOTE CONNECTED (CONDBAT)** specifies the maximum number of inbound connections for database access threads. The default value for this is 10000.

The sum of **MAX USERS** and **MAXDBAT** must not exceed 2000.

The number of data sources should be minimized, as each data source has its own connection pool that is associated with its own set of DB2 threads. The result of having too many data sources is a combination of a higher number of active DBATs and fragmenting the total set of active DBATs.

### 10.2.3 Type 2 or Type 4 JDBC connectivity

Which JDBC connectivity type gives the best performance? Here are some guidelines:

- **When accessing a DB2 database that resides in the same z/OS system image (LPAR) that the application is running on, Type 2 connectivity provides better throughput and CPU Utilization than Type 4 connectivity. This is due to the “network overhead”. The Type 4 driver is implemented using the DRDA protocol over a TCP/IP network connection. Applications accessing local databases residing on the same LPAR using a Type 2 driver do not need to go over a TCP/IP network link. This is true even if the TCP/IP connection does not need to really go out on a physical network link. The fact that you have to go through additional layers (TCP/IP and DDF) introduces overhead that can be avoided when using a local attachment via the Type 2 driver.**
When accessing a DB2 database that resides on a different z/OS system image (LPAR) than the application is running on, Type 4 connectivity provides significant performance improvements over using Type 2 connectivity to a local DB2 system, which in turn uses the DDF DRDA Application requester function to communicate with the remote DB2.

### 10.3 Java Message Service (JMS) and WebSphere MQ

When using JMS applications or Message Driven Beans (MDB) with WebSphere Application Server for z/OS, there are a number of factors that control the throughput of messages and the performance of the application server.

A typical JMS application creates a JMS queue connection from a JMS ConnectionFactory to communicate with the underlying JMS provider. As described earlier, these connections are pooled in a connection pool to provide improved performance. Before the application can access actual queues, it needs to create a queue session from the queue connection. Each queue connection has a pool of sessions associated with it in a session pool, and normally the request will be satisfied with an existing session from this pool, hence improving performance.

A Message Driven Bean (MDB) is associated with a listener port, which acts as a JMS application that delivers messages to the onMessage() method of the MDB. Each listener port is associated with a JMS ConnectionFactory and hence the properties allocated to that factory.

Each active listener port will use one connection, and when a message is delivered on behalf of a listener port, it will use a session from the session pool that is owned by the listener port's connection. The connection used by the listener port will only be returned to the connection pool when the listener port has been stopped. The session used by the listener port is returned to the session pool as soon as the MDB has finished processing the message.

If the Message Driven Bean needs to send a reply message or forward the message on to another JMS destination, then it will require an additional connection and session.

Therefore, there is a sequence of pools (each with their own limits) that affect message throughput and performance. A JMS ConnectionFactory's connection pool should larger than the number of listener ports and other JMS applications that use that connection factory. The session pool associated with a connection pool should be large enough to cover the all of the sessions used by the listener ports and other JMS applications.
### 10.3.1 Message Listener Service

The Message Listener Service in WebSphere Application Server provides the Message Driven Bean (MDB) listening process. The **Listener Ports** define the link to the underlying JMS destination that the MDB wishes to receive messages from.

These Listener Ports are defined within the Message Listener Service.

To view the WebSphere Administrative Console page where you configure this property, select **Servers** $\rightarrow$ **Application servers** $\rightarrow$ `<server_name>` $\rightarrow$ **Messaging** $\rightarrow$ **Message Listener Service** $\rightarrow$ **Listener Ports** $\rightarrow$ `<listener_name>`. This page is shown in Figure 10-4.

![Listener Ports Connection pool](image)

**Figure 10-4** Listener Ports Connection pool

We will now discuss the parameters.
**Maximum sessions**
This value determines how many messages an MDB can process simultaneously.

The default value for this property is 1, which means that an MDB must finish processing the first message on the queue before it can move on to the next one. This means that the MDB is single-threaded.

You can increase the throughput of messages by increasing the value of this property, thus enabling an MDB to process more than one message at once and hence make the MDB multi-threaded.

Therefore, we can see that each active session corresponds to a separate listener thread.

The throughput of an MDB is directly related to the number of MDB instances started by the Listener Port. If this property is set too high, there will be a large number of MDB instances created, which may affect the overall performance of the application server.

**Note:** If you need to process messages in the strict message order that they appear on the underlying message queue, then this value must be set to 1, so only one thread is ever processing messages.

In this case, then you must also set the value of the Maximum messages property to 1.

**Recommendation:** A good starting point if you do not need to maintain strict message order is to use a number equal to 2 * (maximum number of servant regions) * (number of worker threads in one servant).

If your server is configured with the maximum server instances value set to 3, and your Workload profile is set to LONGWAIT, which means that each servant contains 40 worker threads, then Maximum sessions = 2 * 3 * 40 = 240.

**Maximum messages**
This value specifies the maximum number of messages that the listener can process in a single transaction.

If you want to process multiple messages within an MDB in a single transaction, then set this value to more than 1. If messages start accumulating on the queue, then a value greater than 1 enables multiple messages to be batch-processed into a single transaction, and eliminates much of the overhead of transactions on
JMS messages. However, if one message in the batch fails processing with an exception, the entire batch of messages is put back on the queue for processing. Any resource lock held by any of the interactions for the individual messages are held for the duration of the entire batch.

**Recommendation:** Leave this at the default value of 1 and monitor it to see if this meets your needs.

This means that only a single message is put back on the queue in the result of a transaction failure.

**Maximum retries**
This value specifies the maximum number of times that the listener tries to deliver a message to a Message-Driven Bean instance before the listener is stopped.

The default is 0, which means the listener stops immediately it fails to deliver a message.

**Note:** When used in conjunction with WebSphere MQ, it is a good idea to coordinate the value set in this property with the WebSphere MQ queue settings of BOTHRESH and BOQNAME on the underlying queue referenced by the Destination JNDI name entry for this listener port.

- BOTHRESH specifies the backout threshold at which a message that cannot be delivered to an application is removed from the queue and placed into the queue named by BOQNAME.
- BOQNAME is the excessive backout requeue name. This is normally the name of a local queue, although sometimes the dead letter queue may be used.

**Recommendation:** This value should be one greater than the value set in BOTHRESH, as this will allow WebSphere MQ to deal with the undeliverable message and will not stop the listener.
10.3.2 JMS connection factory settings

Here we discuss the JMS connection factory settings.

Connection pool
To view the WebSphere Administrative Console page where you configure this property, select Resources → JMS → JMS providers → WebSphere MQ messaging provider → Queue connection factories → <factory_name> → Connection pool.

This page is shown in Figure 10-5.

![Figure 10-5 WebSphere MQ JMS Provider connection pool](image)

**Maximum connections**
Controls the maximum number of physical connections to the underlying WebSphere MQ Queue manager, as defined in the connection factory.
All other uses of the connection factory will result in additional connections being required from the pool.

**Recommendation:** The maximum size of the connection pool should be larger than the number of Listener ports plus the maximum number of active MDB instances that need a separate connection (Maximum sessions setting on the Listener Ports) plus any other JMS applications that use that connection factory.

**Relationship to WebSphere MQ settings**

For CLIENT mode connections, this must not exceed the MAXCHL Queue Manager attribute on WebSphere MQ for z/OS Version 6. The default value is 200, but a setting of 1000 is recommended for production systems.

For BINDINGS mode connections, this must not exceed the value of the CTHREAD parameter that specifies the total number of threads that can connect to a queue manager on z/OS. The CTHREAD parameter can be found in the CSQ6SYSP macro (see the *WebSphere MQ for z/OS System Setup Guide*, SC34-6583 for details). The default value for CTHREAD is 300.

**Session Pool**

To view the WebSphere Administrative console page where you configure this property, select **Resources → JMS → JMS providers → WebSphere MQ messaging provider → Queue connection factories → <factory_name> → Session pools.**

**Note:** Each MDB Listener Port uses a single connection from this pool. However, if an MDB sends a message during its processing, then an additional connection per active MDB is used from this pool.
Maximum connections
Controls the maximum number of sessions that a connection can have with the underlying WebSphere MQ Queue manager as defined in the Connection Factory.

Recommendation: The session pool associated with a connection pool should be set to the number of worker threads defined by the Application Server Workload Profile.

Other WebSphere MQ settings
There are a number of other WebSphere MQ settings for queue definitions that are advisable when used in conjunction with WebSphere. These are as follows:

- INDXTYPE: This is the type of index maintained by the Queue Manager to speed up message get operations against a queue.
**Recommendation:** Set this value to MSGID.

- **DEFSOPT:** This is the default share option for applications opening this queue for input. The default value for WebSphere MQ on z/OS is EXCL.

**Recommendation:** Set this value to SHARED.

- **SHARE:** This defines whether multiple applications can get messages from this queue simultaneously. The default value for WebSphere MQ on z/OS is NOSHARE.

**Recommendation:** Set this value to SHARED.

### 10.4 CICS Transaction Gateway

*CICS Transaction Gateway for z/OS Version 6.1* provides the J2C compliant resource adapter that is deployed on WebSphere Application Server for z/OS. The resource adapter can be found at Resources → Resource Adapters.
For a detailed description of CICS Transaction Gateway and the way it operates, please refer to *CICS Transaction Gateway for z/OS Version 6.1*, SG24-7161.

CICS Transaction Gateway on z/OS has two different modes of operation: *local* and *remote*. The resource adapter used for this book was connected in local mode only. This means that WebSphere Application Server and the CICS Transaction Gateway are installed in the same Logical Partition (LPAR). It also means that access is via API calls directly into the Gateway; there is no Gateway daemon running and hence no communication connections.

### 10.4.1 CICS Transaction Gateway for z/OS in local mode

**Recommendation:** When accessing CICS systems in local mode, make as many threads available as possible. This can be achieved by using a Workload profile of LONGWAIT, that is, there will be a maximum of 40 threads per servant region. See 7.3.2, “Workload Profile” on page 171 for more details.
To view the WebSphere Administrative console page where you configure this property, select **Servers → Application servers → <server_name> → Container Services → ORB Service → z/OS additional settings → Workload profile**.

This page is shown in Figure 10-8.

![Application servers screenshot](image)

**Figure 10-8 Application Server Workload profile**

**Recommendation**: The RECEIVECOUNT setting in the CICS region for local mode needs to be larger than the total number of EXCI pipes that could try to attach to this CICS region.

For example, if you have one WebSphere Application Server server, one servant, a Workload profile of LONGWAIT (that is, 40 threads), then this should be set to at least 40 as well (there can be a delay in de-allocating pipes, so there can be cases where this will need to be slightly higher than 40. If you have three servants, then there is a potential for a total of 120 EXCI pipes, and in this case this parameter should be set to at least 120.

**Recommendation**: Set the CTG_PIPE_REUSE WebSphere environment variable to the value of ONE.
This is useful if WebSphere connects to multiple CICS regions. This can be used to ensure that a WebSphere Application Server thread will only allocate one EXCI pipe. Without this setting, each thread could allocate one pipe to each CICS region, making it impossible to predict how many EXCI pipes each WebSphere Application Server address space will use.

This variable can be set in the WebSphere Admin Console by selecting Environment → WebSphere Variables.

10.4.2 CICS Transaction Gateway for z/OS in remote mode

Although we did not operate in remote mode, there are a number of different factors to consider here. In a remote configuration, there will be a Gateway Daemon in operation and hence a communications pipe between WebSphere Application Server and the Gateway.

**Recommendation:** The RECEIVECOUNT setting in the CICS region for remote mode needs to match the total of Worker threads in each Gateway that connects to the CICS region.

For example, if there are three Gateways connecting to the same CICS region, the RECEIVECOUNT should be greater than the total of Worker threads in each Gateway.

The Gateway daemon receives network requests from remote Java clients and can handle multiple requests simultaneously. There are two sets of properties in the Gateway configuration file that need attention. There are two pools of threads that can be configured: Connection Manager threads and Worker threads.

For each connected client (such as a connection in the WebSphere connection pool), one Connection Manager thread is used in the Gateway daemon, and is held until the client closes the connection or the Gateway times out on the connection.

**Recommendation:** The Connection Manager threads setting needs to match up with the Connection Pool settings in WebSphere, that is, one connection in the Connection Pool connects to one Connection Manager thread on the Gateway.
In order for a call to CICS, that is, an ECI call, to be performed through an allocated Connection Manager thread, a thread must be allocated from the Worker thread pool for the duration of this request. For this reason, the number of Connection Manager threads can be a lot higher than Worker threads, but there is minimal overhead in having extra Worker threads. There are no monitoring tools to show many Worker threads are actually in use at any point in time.

The Connection Manager threads limit the maximum number of connected Java clients, and the Worker threads the number of concurrent requests that can be issued by these attached clients. The initial and maximum numbers of the Connection Manager and Worker threads are set in the Gateway configuration file using the initconnect, maxconnect, initworker and maxworker parameters. Their default values are set to 1, 100, 1, and 100 respectively.

The same considerations for CTG_PIPE_REUSE apply if you are connecting to multiple CICS regions from one Gateway.

10.5 IMS Connect

IMS Connect Version 2.2 provides the J2C compliant resource adapter, which is deployed on WebSphere Application Server for z/OS. The resource adapter, called the IMS Connector for Java, can be found in the WebSphere Admin Console at Resources → Resource Adapters.
For a detailed description of IMS Connect and the way it operates, please refer to *IMS Connectivity in an On Demand Environment: A Practical Guide to IMS Connectivity*, SG24-6794.

IMS Connect has two different modes of operation: *local* and *remote*. The resource adapter used for this book was connected in both local mode and remote mode.

In local mode, this means that WebSphere Application Server and IMS Connect are installed in the same Logical Partition (LPAR). It also means that access is via API calls directly into the IMS Connect address space and so there are no communications connections.

In remote mode, a TCP/IP connection between IMS Connector for Java and IMS Connect is persistent; in other words, it remains open as long as IMS Connector for Java or IMS Connect do not disconnect it due to an error.
The number of sockets that can exist between a Java client and the host component, IMS Connect, is controlled by two parameters that need to be synchronized. These parameters are MAXSOC in the TCPIP statement of the IMS Connect configuration member and Maximum Connections, a property of the WebSphere resource adapter connection pool.

- **MAXSOC**
  
  The maximum number of sockets for an IMS Connect port. MAXSOC applies to all ports associated with an instance of IMS Connect. The default value for MAXSOC is 50. It can be set to a value between 50 and 65535.
  
  The value of MAXSOC includes one socket dedicated to the Listen State and the remainder are available for connections. For example, if MAXSOC is 50, only 49 can be used by connections from Java clients. If a particular machine has a single instance of IMS Connect with two ports and MAXSOC is set to 50, then each port can have 49 connections, making a total of 98 connections into the IMS Connect instance.

- **Maximum Connections**
  
  There is a direct correlation between the Maximum Connections property in the resource adapter connection pool setting in WebSphere and the value of MAXSOC in IMS Connect.

  **Recommendation:** Maximum Connections in the connection pool should be set a value less than MAXSOC.

### 10.6 Using the Performance Monitoring Infrastructure (PMI)

The Performance Monitoring Infrastructure (PMI) collects performance data for both J2C Connectors and JDBC drivers. These connection pool counters provide valuable information for monitoring the performance of the connectors and can give an early warning where things may be going wrong.
Table 10-1 shows the counters that are available for J2C connectors, and indicates the level in the PMI statistic set where the collection of this information is turned on.

Table 10-1  PMI counters for J2C

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>PMI Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>ManagedConnectionCount</td>
<td>The number of ManagedConnection objects in use.</td>
<td>Custom</td>
</tr>
<tr>
<td>ConnectionHandleCount</td>
<td>The number of connections that are associated with ManagedConnections (physical connections) objects in this pool.</td>
<td>Custom</td>
</tr>
<tr>
<td>CreateCount</td>
<td>The total number of managed connections created.</td>
<td>Basic</td>
</tr>
<tr>
<td>CloseCount</td>
<td>The total number of managed connections destroyed.</td>
<td>Basic</td>
</tr>
<tr>
<td>AllocateCount</td>
<td>The total number of times that a managed connection is allocated to a client (the total is maintained across the pool, not per connection).</td>
<td>Custom</td>
</tr>
<tr>
<td>FreedCount</td>
<td>The total number of times that a managed connection is released back to the pool (the total is maintained across the pool, not per connection).</td>
<td>Custom</td>
</tr>
<tr>
<td>FaultCount</td>
<td>The number of faults, such as timeouts, in the connection pool.</td>
<td>Custom</td>
</tr>
<tr>
<td>FreePoolSize</td>
<td>The number of free connections in the pool.</td>
<td>Basic</td>
</tr>
<tr>
<td>PoolSize</td>
<td>Average number of managed connections in the pool.</td>
<td>Basic</td>
</tr>
<tr>
<td>WaitingThreadCount</td>
<td>Average number of threads concurrently waiting for a connection.</td>
<td>Basic</td>
</tr>
<tr>
<td>PercentUsed</td>
<td>Average percent of the pool that is in use.</td>
<td>Custom</td>
</tr>
<tr>
<td>PercentMaxed</td>
<td>Average percent of the time that all connections are in use.</td>
<td>Custom</td>
</tr>
<tr>
<td>WaitTime</td>
<td>Average waiting time in milliseconds until a connection is granted.</td>
<td>Basic</td>
</tr>
<tr>
<td>UseTime</td>
<td>Average time in milliseconds that connections are in use.</td>
<td>Basic</td>
</tr>
</tbody>
</table>
Table 10-2 shows the counters that are available for JDBC, and indicates the level in the PMI statistic set where the collection of this information is turned on.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>PMI Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrepStmtCacheDiscardCount</td>
<td>The total number of statements discarded by the least recently used (LRU) algorithm of the statement cache.</td>
<td>Extended</td>
</tr>
<tr>
<td>JDBCTime</td>
<td>The amount of time in milliseconds spent running in the JDBC driver (includes time spent in the JDBC driver, network, and database).</td>
<td>Extended</td>
</tr>
<tr>
<td>ManagedConnectionCount</td>
<td>The number of ManagedConnection objects in use.</td>
<td>Custom</td>
</tr>
<tr>
<td>ConnectionHandleCount</td>
<td>The number of connections that are associated with ManagedConnections (physical connections) objects in this pool.</td>
<td>Custom</td>
</tr>
<tr>
<td>CreateCount</td>
<td>The total number of managed connections created.</td>
<td>Basic</td>
</tr>
<tr>
<td>CloseCount</td>
<td>The total number of managed connections destroyed.</td>
<td>Basic</td>
</tr>
<tr>
<td>AllocateCount</td>
<td>The total number of times that a managed connection is allocated to a client (the total is maintained across the pool, not per connection).</td>
<td>Custom</td>
</tr>
<tr>
<td>FreedCount</td>
<td>The total number of times that a managed connection is released back to the pool (the total is maintained across the pool, not per connection).</td>
<td>Custom</td>
</tr>
<tr>
<td>FaultCount</td>
<td>The number of faults, such as timeouts, in the connection pool.</td>
<td>Extended</td>
</tr>
<tr>
<td>FreePoolSize</td>
<td>The number of free connections in the pool.</td>
<td>Basic</td>
</tr>
<tr>
<td>PoolSize</td>
<td>Average number of managed connections in the pool.</td>
<td>Basic</td>
</tr>
<tr>
<td>WaitingThreadCount</td>
<td>Average number of threads concurrently waiting for a connection.</td>
<td>Basic</td>
</tr>
<tr>
<td>PercentUsed</td>
<td>Average percent of the pool that is in use.</td>
<td>Custom</td>
</tr>
<tr>
<td>PercentMaxed</td>
<td>Average percent of the time that all connections are in use.</td>
<td>Custom</td>
</tr>
<tr>
<td>WaitTime</td>
<td>Average waiting time in milliseconds until a connection is granted.</td>
<td>Basic</td>
</tr>
</tbody>
</table>
These counters can be viewed in a number of monitoring tools, including the Tivoli Performance Viewer (TPV) and IBM Tivoli Composite Application Manager (ITCAM). See “IBM Tivoli Performance Viewer (TPV)” on page 96 and 5.2.4, “IBM Tivoli Composite Application Manager” on page 109 for more information about those tools.

The information collected at the "Basic" level is ideal for general monitoring purposes. It shows you how many connections are currently in the pool (PoolSize and FreePoolSize) and the average number of threads waiting for a connection (WaitingThreadCount). It can also tell you the average time that a thread has to wait to obtain a connection (WaitTime). If the pool is full and there are a large number of waiting threads, then you need to consider increasing the size of the pool within the limitations imposed by the back-end systems. By examining the amount of time that a connection is actually used, you might be able to identify performance problems in the processing that takes place in the back-end system. If this time is excessive, then there might be a problem.

For JDBC Connectors, some additional JDBC specific information can be obtained at the "Extended" level. There is no additional information for J2C Connectors at this level.

Additional information can be obtained at the "Custom" level, but this is normally used when trying to identify a specific performance issue. For example, you will look carefully at PercentMaxed to see how often all of the connections in a pool were in use. If this value is consistently high, then you should consider increasing the maximum size of the pool.

### 10.7 References

- CICS Transaction Gateway for z/OS Version 6.1, SG24-7161
- IMS Connectivity in an On Demand Environment: A Practical Guide to IMS Connectivity, SG24-6794
- WebSphere for z/OS Connectivity Architectural Choices, SG24-6365
- WebSphere for z/OS V6 Connectivity Handbook, SG24-7064
- DB2 for z/OS and WebSphere: The Perfect Couple, SG24-6319
- WebSphere for z/OS to CICS and IMS Connectivity Performance, REDP-3959
Best practices - application

We talk in Part 2, “Best practices - infrastructure” on page 143 extensively about the most important parameters and settings in the infrastructure, that is, WebSphere Application Server, the Java Virtual Machine and the underlying subsystems. Following this information should help you considerably in preparing an environment that is optimized for performance. However, this is only one part of being successful. The performance of a WebSphere environment depends largely on how well the application has been developed. Most of the application best practices are not z/OS specific, but a “bad” practice may have a bigger impact on performance in an environment that is heavily shared, such as z/OS, than an environment that is only dedicated to that one (bad) application, such as a stand-alone Windows server. Also, WebSphere servers on stand-alone servers tend to be recycled more easily to solve problems; which is something that is not easily done on z/OS! The bottom line is that J2EE applications for z/OS should be developed carefully while keeping best practices in mind. This part of the book is an attempt to list and describe those best practices. Use them as a checklist for J2EE applications being deployed to WebSphere Application Server on z/OS.
We have divided this part into two chapters:

- Chapter 11, “J2EE application best practices” on page 269 describes best practices for the application itself.
- Chapter 12, “Accessing EIS resources” on page 301 describes application development best practices for connecting to back-end systems.
J2EE application best practices

Performance often determines the success or failure of enterprise applications. Every application is different and exhibits different behavior under a variety of conditions. Therefore, this chapter provides a guide to achieve optimal performance in your application code and best practices associated with common Java EE components used in a large proportion of applications.
11.1 Introduction to performance of J2EE applications

A multi-tiered application typically depends on a large number of tunable parameters for improving the performance of the system. Many of the tunable parameters are useful only for applications with certain workload characteristics; a known set of configuration parameters optimized for one workload may not be the best for another.

This chapter walks through the optimization of a J2EE application in order to illustrate best practices for developers in order to obtain high performance.

11.1.1 Terminology

A full evaluation of the performance of an application often requires the use of different metrics and different perspectives.

Initially, it is usually best to put the focus at the transaction level and measure the individual transaction time or page response time. After that, the focus expands to include measurements of the transaction throughput, the ability to support concurrent users, and the overall scalability of the application. One of the challenges in terms of performance in application development is to try to balance these points. Here are a few definitions:

Response time
The time used for an application to process a request, for example, an HTTP request from a user's browser. Normally, we are most interested in average response time, although it is more important to consider consistency of response times, even under peak loads. Spikes in response time are often an indicator of poor behavior under peak loads, and potential instability.

Latency
The minimum time taken to get a response from the application, despite whether the application has to do much work to produce the response.

Throughput
The amount of work that can be performed by an application or component in a given period of time.

A Web site can only handle a specific number of concurrent requests. Throughput depends on that number and on the average time a request takes to process; it is measured in requests per second. If the site can handle 100 concurrent requests and the average response time is one second, the Web site's throughput is 100 requests per second.
Scalability  Refers to how the application can respond with changing volumes, typically, a higher concurrent load, often resulting from a larger user community.

11.1.2 Approach to improve performance

First, it is necessary to highlight the importance of taking an evidence-based approach to performance analysis.

It is essential that you have concrete data about the real application performance, rather than suppositions or prejudices, before starting your optimization work. This will avoid unnecessary costs and dangers related to too many code modifications.

To obtain this data, we suggest executing different load tests using workload driver tools, in an environment with the same software and hardware available as the production environment where the application will finally run. Some workload driver tools are discussed in Chapter 4, “Using workload drivers” on page 33.

It is important that the tests are repeatable, and that their methodology is documented. Be sure to avoid common errors, such as:

- Do not trust the result of one execution. Make sure that it is repeated.
- Be sure that the load testing software is not distorting the results.
- Be sure that the loading test software is not competing with the application server running the application for CPU time. This is a risk if this software runs on the same system or system image as the actual application. For Web applications, it is best to run load tests from one or more other physical machines or system images that are not sharing the same CPU, to eliminate this effect.

If you have a performance problem, it is a good idea to make changes to the application that significantly improve performance. However, it is not a good idea to make the wrong changes!
Also, it is very easy to make changes that “theoretically” should improve performance but never seem to make a difference. It is important to follow the steps specified below in order:

1. Set a clear goal.

   It is essential to have clear goals for performance and scalability. Those goals should be more accurate and better quantified than “it should run fast” or “it should be highly scalable”. Some of the key questions are:
   - What are the throughput and response time goals? Without clear goals, it is impossible to know whether performance is satisfactory.
   - Which operations must be fast, and which can be slow? Not all use cases are equal. Some must be prioritized from a performance perspective.
   - What is the maximum number of concurrent users for the application? Without this information, it is impossible to verify success.

2. Identify problem areas.

   It is important to identify the bottlenecks when you start making changes to improve performance. A short investigation of problems might reveal the specific component that causes poor performance.

3. Follow a methodical and focused path.

   Once the goal is set, try to make changes that are expected to have the biggest impact on performance, that is, go for the “low hanging fruit” first. Your time is better spent tuning a method that takes 10 seconds but gets called 100 times than tuning a method that takes one minute but gets called only once. You can make one change at a time and load test it. If the change results in positive impact, only then will you make it permanent.

### 11.1.3 Infrastructure tuning

In general, it is best to explore infrastructure tuning options before getting into the application code analysis. Some parameters and settings can be very easily checked in the z/OS runtime environment and may already create a satisfactory performance. We discuss a large number of those infrastructure parameters and settings in Part 2, “Best practices - infrastructure” on page 143. You should remember that optimizing the application may take more time than just checking those parameters and settings on z/OS.

We recommend reviewing:

- Chapter 7, “WebSphere Application Server and HTTP Server” on page 159
- Chapter 8, “Java Virtual Machine” on page 191
- Chapter 9, “z/OS subsystems” on page 217
- Chapter 10, “Connectors” on page 237
Database
The database can be a major enterprise-level bottleneck, and for this reason it is critical that your database is correctly configured. Database optimization can be complex and vendor dependent, and the database administrator (DBA) collaboration is important to succeed.

It is outside the scope of this book to discuss back-end system tuning, but we find it important to mention it.

Settings differ between databases, but important settings include:
▶ Threading models
▶ Listener pools
▶ Indexes
▶ Table structure
▶ Query optimization options

Database products usually provide profiling tools that can help to indicate potential performance improvements.

11.2 Application profiling

The answer to the question "What do I do when my load testing gives me performance numbers that are not good enough?" lies in profiling.

Two steps lead to better performance through profiling and they must be performed iteratively:
▶ Load testing
▶ Profiling (includes tuning, based on information obtained during profiling)

The profiler analysis helps you to spot performance bottlenecks. Execution times of different methods, monitor contentions, and object allocations are important figures to look at during profiling. The aim of profiling is to make sure that relatively high amounts of time are not being spent in a certain part of the code.

Knowledge of the time spent in certain methods may require rewriting a method. Adherence to good Java programming practices also helps.

The famous 80/20 rule also applies here; typically, 80% of a code's execution time will occur in 20% of the code. For this reason, it is important to focus performance analysis on the areas that make the biggest difference.
You should start your tuning by focusing on areas that you have measured that are worse than the criteria that you set in the planning phase. Of these, your most critical scenarios take precedence.

After you have targeted a scenario, profile the scenario with a profiler, and focus on the methods that take the most time to complete.

Work through the methods where you can make the biggest difference until you meet the performance criteria. Then, move to the next scenario.

We used the open source Eclipse Test & Performance Tools Platform (TPTP) and IBM Tivoli Composite Application Manager (ITCAM) for application profiling. For more details about those tools, refer to 5.2.2, “Eclipse TPTP” on page 71 and 5.2.4, “IBM Tivoli Composite Application Manager” on page 109 for more details.

11.3 J2EE overview

The Java 2 Platform, Enterprise Edition (J2EE) is a specification for defining software components and services. Based on these modular components distributed multi-tier architecture Java applications are developed.

J2EE components require a so called J2EE server as their runtime environment. In our case, this is WebSphere Application Server, running on z/OS.

The client-side access to a J2EE application in most cases happens through a browser. It is also possible to access J2EE applications on the server through a “fat” client, for example, a Java client-side application, a CORBA client-side application, a WebSphere MQ client, or a Web services client.

This J2EE server is divided into several logical components, so called containers. The actual specification requires the following containers:

- **EJB container** as runtime environment for Enterprise JavaBeans
- **Web container** as runtime environment for JavaServer Pages and servlets
- **Client container** as runtime environment for application clients and applets
An overview is provided in Figure 11-1.

**Figure 11-1  J2EE overview**

### 11.3.1 JavaServer Pages (JSP)

*JavaServer Pages (JSP)* is a technology that generates HTML or XML output of a Web server dynamically as a response to a Web client request.

It allows to embed Java code or special XML tags, called JSP actions, in static content. The advantage is that logic and design can be implemented independently from each other.

The JSPs are translated into Java servlets by a JSP compiler. These servlets are then executed in the Web container.

### 11.3.2 Servlets

*Servlets* are Java classes. Their instances process and answer requests from clients in a Web server. This technology allows you to generate Web content dynamically. The instances of a servlet are created by the Web container.

### 11.3.3 Enterprise Java Beans (EJB)

*Enterprise Java Beans* are standard components in the J2EE server. They simplify the development of distributed multi-tier architecture Java applications. They provide concepts for enterprise applications which are required for the business logic of an application.
There are different types of EJBs, as we see in the next three sections.

**Entity Beans**

*Entity Beans* query and update the persistent data of a system. They represent the data as stored in a database. The way the persistence is taking place can be influenced by the developer of the Entity Bean. The behaviour of the Entity Bean is determined by the settings in the Deployment Descriptor of the Bean and additional settings or rules at deployment time in the WebSphere Admin Console.

**Session Beans**

*Session Beans* represent interactions between user and system. They often use several Entity Beans to persist data being changed or added. We can distinguish between *stateful* and stateless Session Beans:

**Stateful Session Bean**

A stateful session Bean can save information for later use. This means that when an application invocation consists of multiple requests in a certain time period, follow-on requests can reuse saved data from the earlier requests. Stateful session Beans are identified by their their unique (stem generated) ID.

**Stateless session Bean**

A stateful session Bean cannot save information, which means that all information required for a method invocation has to be handed over as a parameter to the next component. We cannot distinguish between different instances of a stateless session Beans, as they do not have an ID.

**Message Driven Beans (MDB)**

*Message Driven Beans (MDBs)* are the components that make EJB systems accessible for asynchronous communications. Therefore, the Java Message Service (JMS) is used. For more information about MDBs on z/OS, refer to *Java Message Service (JMS) Security on z/OS*, REDP-4203.

**11.3.4 Web services**

A *Web service* is a software component with a unique ID, so called *Uniform Resource Identifier (URI)* and interfaces written in XML. It allows for direct communication with other software components using XML based messages through the exchange of Internet based protocols.

There are many books and papers on Web services and we do not intend to explain the architecture in this book. Figure 11-2 shows a high-level diagram.
11.3.5 Other J2EE APIs

J2EE is much more than just servlets, JSPs, EJBs and Web services. There are numerous APIs included in J2EE now and the specification is still changing and expanding. Table 11-1 shows the most important J2EE APIs.

Table 11-1 Some other important J2EE APIs

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Database Connectivity (JDBC)</td>
<td>This API allows an application to access a relational database using SQL.</td>
</tr>
<tr>
<td>Java 2 EE Connector Architecture (J2C)</td>
<td>This API allows the transparent integration of other systems (Enterprise Application Integration).</td>
</tr>
<tr>
<td>Java Message Service (JMS)</td>
<td>This API defines a standard for connecting the Java EE to heterogeneous Enterprise Information Systems (EIS), such as CICS and IMS.</td>
</tr>
<tr>
<td>Java API for XML Processing (JAXP)</td>
<td>This API is a facility to process XML documents.</td>
</tr>
<tr>
<td>Java API for XML based Remote Procedure Calls (JAX-RPC)</td>
<td>This API allows remote access to RPC services.</td>
</tr>
</tbody>
</table>

Note: There are many more APIs provided by Java 2 EE, but an explanation of all of them is beyond the scope of this book.
11.3.6 J2EE and performance

Whichever components you choose to use and whichever APIs you include in your code, your final application will consist of a number of Java classes and XML files. All of those components will be used by the JVMs running in the servant regions of WebSphere Application Server. From a performance perspective, all components (classes and files) deployed to WebSphere Application Server are part of the scope of performance analysis. The tools discussed earlier in this book apply to everything running in the JVMs of the application server, so each component can be tracked down and investigated.

It is good to have background knowledge of the J2EE model when performing application profiling. It is important to know which code you have written yourself, which code is generated, and which code is reused from other sources. Be aware that you may be using (inefficient) Java classes in your application that you cannot change, for example, do not own the particular source.

11.4 Application best practices

In the following sections, we discuss the most important best practices applying to a J2EE application. It is not a definite list, but it does contain the ones that can make a big difference.

11.4.1 Java coding best practices

There are many Java programming optimization techniques, so we do want to attempt to enumerate them here. Instead, we will talk about optimizations that can produce results very easily.

**Algorithms**

Before starting with low-level code details, examine the algorithm used by a slow method. Ask the following questions:

- Is there any redundancy?
- Are there any missed opportunities for caching?
- Is there a better algorithm for this problem?
- Are there too many objects created?

**Collection types**

Choosing the correct *collection type* has an effect on performance. For example, if you need to read in a large number of elements, pre-allocating a large ArrayList is faster than using a LinkedList. And random access to an ArrayList is faster than random access to a LinkedList.
It is easy to see why this is true by looking at the respective implementations in the java.util package.

**String operations**
Avoid unnecessary String operations, because string concatenation in Java is slow. Since strings are immutable, a String cannot be modified, so string concatenation always results in at least one new object. See Example 11-1.

**Example 11-1  String concatenation use**

```java
String strData = new String("");
for (Iterator it = results.iterator(); it.hasNext();)
{
    String account = (String) it.next();
    strData += account;
}
```

Often, using a StringBuffer is more efficient than using a String.

Also, use StringBuffer rather than the + operator to perform concatenations that involve runtime-dependent string variables. See Example 11-2.

**Example 11-2  StringBuffer implementation**

```java
StringBuffer sbData = new StringBuffer ("");
for (Iterator it = results.iterator() ; it.hasNext() ;)
{
    String account = (String) it.next();
    sbData.append(account);
}
```

**Logging**
Because String operations are slow, and are often written without any concern as to efficiency (displaying object information, for example), toString() calls are often slow.
Also, avoid the System.out.println or System.err.println calls. These calls are often used for debugging and tracing and can be troublesome when you want migrate code from the development environment to the testing or production environment. The logging calls also slow down the performance because I/O, which is a synchronized operation, needs to be performed.

You should try to use a logging mechanism (such as log4j) that lets you switch off logging in the production environment to reduce logging overhead. Another advantage of a good logging framework can be that the actual I/O is performed asynchronously.

**Exceptions/Error managing**

Develop a consistent approach for managing application exceptions and business errors. Use a mechanism, such as the error list utility, to manage a list of exceptions and errors to communicate them back to clients in a standard way. A standard application exception can be used to communicate a list of errors back to a client. The error list utility should be integrated into the transaction management service to ensure data integrity. See Example 11-3.

```java
public ArrayList getProducts (int prodCod) throws Exception
{
    ArrayList products  = new ArrayList();
    try {
        Iterator prods = getProdHome().findProduct(new Integer(prodCod)).iterator();
        while (prods.hasNext())
        {
            ProductLocal prodLoc  = (ProductLocal) prods.next();
            products.add(getProdModel(prodLoc));
        }
        return products;
    }
    catch (Exception e) {
        e.printStackTrace();
        throw fe;
    }
}
```

Avoid the printStackTrace() calls in each method where the exception throws, and centralize only the necessary exception or error data in the standard application exception, as shown in Example 11-3.
Example 11-4 shows a more customized way of throwing exceptions.

Example 11-4  Custom business exception use

```java
public ArrayList getProducts (int prodCod) throws businessException
{
    ArrayList products = new ArrayList();
    try {
        Iterator prods = getProdHome().findProduct(new Integer(prodCod)).iterator();
        while (prods.hasNext())
        {
            ProductLocal prodLoc = (ProductLocal) prods.next();
            products.add(getProdModel(prodLoc));
        }

        return products;
    } catch (FinderException fe) {
        throw ExceptionFactory.createBusinessException(ExceptionFactory.ACTION_FINDER,
                ExceptionFactory.MODEL_KEY_PROD, prodCod, fe)
    }
}

public class businessException extends Exception implements exceptionKeys
{
    int action     = ACTION_NONE;
    int modelKey   = MODEL_KEY_NONE;
    int modelID    = 0;
    Exception involvedException;

    public BusinessException()
    {
        super();
    }

    public BusinessException(int newAction, int newModelKey, int newModelID, Exception newInvolvedException)
    {
        super();
        action     = newAction;
        modelKey   = newModelKey;
        modelID    = newModelID;
        involvedException = newInvolvedException;
    }
```
Aggregate business objects
Applications often make multiple calls and return multiple values. In order to do that, it is good practice to aggregate data use, for both input parameters and returned data. Instead of calling a method that takes both an address and telephone number, create a method that takes a person. This improvement requires only a single object, so by aggregating data, fewer method calls need to occur and a greater amount of data can be returned in a single call.

This approach is named coarse grained and the objective is minimizing the number of network calls to improve performance.

Data structure
The choice of data structures includes:
- Individual Java data types and objects
- Value objects
- XML data
- Collections

None of them are necessarily right or wrong in any situation. Choose the generic data structure that best fits your needs, XML data is the most flexible choice for automated components. However, it performs the worst, especially for large amounts of data.

For many cases, a collection of value objects that implement a standard value object interface works well. For simpler architectures that do not involve many hierarchies or collections of objects, an argument list containing name and value pairs can also work fine.

Data Caching
The appropriate use of caching can produce large performance gains. Caching can be complex and error-prone, and it is best not to implement any caching without proof that there is a strong reason to do it.

Caching is possible in all architectural tiers. It is often best to avoid duplication of caches, and consider the whole application stack, rather than focus on caching in a particular tier.

Two primary things to consider are the amount of memory that is consumed in order to cache these objects, and the rate at which the data changes. As the
entire cache mechanism consumes more and more memory, there is less memory available for application objects, which can also slow an application down, because the JVM will be forced to collect garbage more often. The other issue to consider is that frequently changed data will require a large amount of effort to keep up to date in the cache, which will also consume resources.

Good candidates for cache data exhibit the following characteristics:

- Fairly static or read-only data
- Used frequently or predictably by the application
- Small volumes of data

Application components obtain objects such as RDBMS DataSource, JMS ConnectionFactory, JMS Queue, and the EJB Home object by performing a Java Naming and Directory Interface™ (JNDI) API lookup. You can reduce the overhead of looking up these objects by caching and placing the objects in a common accessible place. Lookup operations can check in the common repository before performing the JNDI API lookup.

By caching this lookup, these services can be obtained more quickly and with fewer resources. The J2EE design pattern Service Locator, implements this technique using a class to cache service objects, methods for JNDI lookup, and methods for getting service objects from the cache.

**Asynchronous processing**

Asynchronous processing is a strategy that can be used in certain circumstances to alleviate performance concerns. There are a limited number of situations for which this approach can be used; however, in the cases in which it is applicable, it can make a clear difference.

Use asynchronous processing in cases in which a certain user request would spend an excessive amount of time processing the request and you want an immediate response or feedback to be sent to the user. You can use Java Message Service (JMS) and Message-Driven Beans (MDBs) to provide asynchronous processing of messages and requests.

**XML parsing**

Converting Java objects to and from XML is usually very expensive. XML data binding can reduce the cost, but not eliminate it.

If an application does a large amount of XML parsing, it is important to look at the parsing method used to do it. Two of the basic parsing options are the Document Object Model (DOM) and the Simple API for XML (SAX).
DOM parsers require much more overhead, because they parse an entire XML document and create a memory object representation of the XML tree. This is helpful if the program requires either significant manipulation or the creation of XML documents.

However, if your application simply needs to parse through a document once and obtain the data, the SAX parser is much more efficient. It reads through a document once and invokes methods to process each tag that it comes across in the document.

Note: Please refer this technical document to obtain more details about the right XML processing for your applications:

Object serialization
Objects are serialized to be stored in files or to be sent over a network. When an object instance is serialized, its instance fields are also serialized.

Serialization is a recursive process and can be costly if it involves a large number of objects containing deep references to other objects. Mark as transient those fields that need not be serialized or that can be manually generated. Doing so reduces the overhead of serialization, because not all fields have to be serialized in an object instance.

Variables that have access modifier transient will not be read from or written into streams. It improves the performance by avoiding writing unnecessary data into streams.

11.4.2 Web Container best practices

Besides the infrastructure-related settings discussed in 7.3.13, “Web container” on page 186, the following application-related best practices could be important. These best practices apply to the Java components running in the Web container of the application server.

Keep the HTTP session size to a minimum
A large session size can quickly degrade the scalability and performance of applications. We recommend that the size of the session be kept as small as possible. Exactly how small depends on the characteristics of your particular application, but in general, use the session to store a minimal amount of state needed to maintain future operations.
Remember that JSP files require participation in an HTTP session by default, and if you do not use HTTP session in your JSP files, you can avoid the overhead involved related to session management with the following JSP page directive:

```
<%@ page session="false"%>
```

In order to reduce memory overhead and to improve performance, it is better to remove/invalidate session explicitly using the session.invalidate() method, rather than the Web container ending the session after a certain timeout (that may be generic for the entire application server environment).
Figure 11-3 and Figure 11-4 on page 287 display a bad practice related to the user session objects utilization. The session size increased from 62457 bytes to 92925 bytes in six minutes of site navigation. This info refers to only one user. For this reason, it is very important to be sure to explicitly remove objects from the session scope when you no longer require the session’s access to these objects.

**Figure 11-3  User session object - Time 5:54:07 PM**
Use appropriate scope for bound objects

The JSP specification supports four levels of scope (application, session, request, and page) that you can associate with objects created explicitly for JSPs to use. Because objects bound to any of these four scopes consume memory and require cleanup, it is best to use the appropriate scope for the task you want to do.
Example 11-5 shows an example of the scope in a JSP.

**Example 11-5  JSP scope use**

```html
<%@ page errorPage="errorpage.jsp" %>

<jsp:useBean id="user" scope="request" class="beans.user" />

<html>
  <head>
    <title>Request Bean Example</title>
  </head>
  <body>
    You entered
    Name: <%= user.getUsername() %>
    Email: <%= user.getEmail() %>
    Age: <%= user.getAge() %>
  </body>
</html>
```

In general, you can begin with a request scope and then evaluate whether you need more scope for your created objects.

**Choose the right include mechanism**

Static data, such as headers, footers, and navigation bar content, is best kept in separate files and not regenerated dynamically. Once such content is in separate files, they can be included in all pages using one of the following include mechanisms:

- **Include directive**
  ```jsp
  <%@ include file="filename" %>
  ```

- **Include action:**
  ```jsp
  <jsp:include page="page.jsp" flush="true" />
  ```

The first include mechanism includes the content of the specified file while the JSP page is converted to a servlet (translation phase), and the second include includes the response generated after the specified page is executed.

Use the include directive, which is fast in terms of performance, if the file does not change often; and use the include action for content that changes often.
Implement valuable JSP custom tags

Custom tags in JSP give you reusability and simplicity. Simplicity means that you need not write Java code in a JSP; rather, you write custom tags. Reusability means that once you write a piece of code as custom tag handler, you can use this tag handler in any JSP, so you can change the entire set of presentation pages without modifying each page individually.

But what will happen if you write a tag handler that is not reused often? In such cases, it is better not to use custom tags, since you need to use classes, interfaces, deployment descriptor files, and also you need to override methods of those classes and interfaces in order to write a tag handler. The JSP engine has to look at the descriptor file to figure out the tag handler class and execute that handler. All these operations do not come for free. It reduces performance and it is proportional to the number of tag handlers you use in JSP. So, do not use custom tags unless you are sure of its reusability and for the same reasons, the excessive use of custom tags also may result in poor performance.
Use servlet caching options
The default mechanism of a servlet engine is to load a servlet in a multithreaded environment. In this environment, a servlet init() method is called only once in its lifetime. You can improve performance using this method to cache static data and execute expensive operations that need only be performed during initialization. An example is shown in Example 11-6.

The destroy() method is called only once in its servlet lifetime, when the servlet engine is removed from memory, so you can use it to remove instance variable resources to avoid memory leaks.

Example 11-6  Servlet init() method as cache

```java
public void init()
{
 String requestMappingsURL = null;
 try
 { 
     requestMappingsURL =
     getServletContext().getResource("WEB-INF/xml/requestmappings.xml").toString();
 } catch (java.net.MalformedURLException ex)
 { 
     System.out.println("ScreenFlowManager: initializing ScreenFlowManager malformed URL exception: " + ex);
 }

 urlMappings = ScreenFlowXmlDAO.loadRequestMappings(requestMappingsURL);
 getServletContext().setAttribute(WebKeys.URLMappingsKey, urlMappings);
}
```

Disable JSP auto reloading
The JSP engine has the capability of loading JSP pages dynamically, which means you need not restart your application server whenever you change the JSP content.

When the JSP auto reloading is switched on, the application server JSP engine loads the JSP's servlet every time when you configure that JSP's servlet. For example, if you configure auto reload time as one second, then the JSP engine loads that JSP's servlet after every one second.

This feature is good because it reduces the development time by avoiding a restart of the server after every change in a JSP. But it gives poor performance in the production environment due to unnecessary loading on class loader. So, turn off your auto reloading feature in the configuration file to improve performance.
Abstract HTTP request
Abstract the HTTP request as a user event to isolate the specific protocol used from the front-end logic. The user event can then be used to drive the other key abstractions in the controller architecture. These include the action, the service, and the next Web page. By abstracting these key elements of user interaction, a good portion of the front-end processing can be automated and defined through metadata, such as an XML file, for example.

Avoid hardcoded physical URLs in the application
Abstract physical URLs out of the application into metadata in order to enable automation, avoid broken links, and have more maintainable pages. The controller architecture can use the metadata to resolve logical page names and forward control to the corresponding physical URL, typically implemented by a JSP page.

WebSphere Web container settings
To check the available settings and best practices for the Web container, please review 7.3.13, “Web container” on page 186.

11.4.3 EJB Container best practices
Besides the infrastructure-related settings discussed in 7.3.14, “EJB container” on page 189, the following application-related best practices could be important. These best practices apply to the Java components running in the EJB container of the application server.

Use EJB local interfaces wherever possible
To avoid overhead, you should first determine if a distributed call is needed. If a local Java object can handle the entire request, and no real business requirements mandate a remote call, then the call should be local. If the serving object does not need to make use of container security, instance pooling, container managed transactions, and the object does not need to be distributed, do not use an EJB.
When you definitely need an EJB, make use of local interfaces where practical. Since EJB 2.0, local interfaces allow non-marshalled object invocations, allowing efficient pass-by-reference parameters and reducing overhead. For these reasons, you can make a local call with much better performance.

Figure 11-5 and Figure 11-6 on page 293 show an example of the same application logic executed with a remote interface and a local interface respectively.

Figure 11-5  getEstuCons() method using EJB Remote interface - cum. time: 8.16 seconds
Evaluate direct JDBC calls

Do not use Entity Bean finder methods to iterate through a large result set of business objects. For a result set of n objects using lazy loading, this can actually cause (n + 1) database interactions.

Use a JDBC wrapper component to run the database query and hold the result set for iteration or return a set of value objects. This limits the operation to one database interaction.
If transactional update operations are required, a business object can be instantiated from a given row in the result set. See Figure 11-7 and Figure 11-8 on page 295.

Figure 11-7  getEstuPaci() method using local EJB - cum. time: 24.06 seconds
For read-only access to a set of data that does not change frequently, use the "Fast Lane Reader" pattern, which bypasses the EJBs and uses a data access object that encapsulates access to the data.

**Note:** For more details about this pattern, please refer to:

http://java.sun.com/blueprints/patterns/FastLaneReader.html
Review EJB CMP 2.x improvements
For Enterprise JavaBeans (EJB) 2.x CMP entity beans, you can use new features related to the database data representation in order to improve performance. We recommend reviewing these sections in the WebSphere Application Server Infocenter:

- Batched commands for container managed persistence

- Deferred create for container managed persistence

- Partial column updates for container managed persistence

- Explicit invalidation in the Persistence Manager cache

Wrap entity beans with a session bean
Avoid accessing EJB entity beans directly from JSP or servlet code. Instead, wrap and access EJB entity beans in EJB session beans. This best practice satisfies two performance concerns:

- Reducing the number of network calls. When the client application accesses the entity bean directly, each getter method is a network call. A wrapping session bean can access the entity bean locally, and collect the data in a structure, which it returns by value, in one network call.

- Providing an outer transaction context for the EJB entity bean. When the session bean wraps the entity bean to provide an outer transaction context, the entity bean synchronizes its state when the outer session bean reaches a transaction boundary.

Check transaction settings
You need to control EJB transaction settings to avoid unnecessary transaction propagation on every method. You can divide a bean's methods into transactional methods and non-transactional methods, and assign transaction attributes only to transactional methods, assign “NotSupported” or “Never” to non-transactional methods so that you avoid transaction propagation. But note that “NotSupported” or “Never” attributes cannot be used for entity beans,
because entity beans need to be involved in a transaction to commit data, so use these attributes for session bean's non-transactional methods only.

In Example 11-7, all the BusinessAccount EJB methods require a transaction, and in Figure 11-8 on page 295, only the method BusinessAccount requires a transaction state.

Example 11-7  EJB Deployment Descriptor - Assembly descriptor section by default

```xml
<assembly-descriptor>
  <container-transaction>
    <method>
      <ejb-name>BusinessAccount</ejb-name>
      <method-name>*</method-name>
    </method>
    <trans-attribute>Required</trans-attribute>
  </container-transaction>
</assembly-descriptor>
```

Example 11-8  Specific transaction attribute for each method

```xml
<assembly-descriptor>
  <container-transaction>
    <method>
      <ejb-name>BusinessAccount</ejb-name>
      <method-intf>Local</method-intf>
      <method-name>setAccount</method-name>
      <method-params>
        <method-param>int</method-param>
      </method-params>
    </method>
    <trans-attribute>Required</trans-attribute>
  </container-transaction>
  <container-transaction>
    <method>
      <ejb-name>BusinessAccount</ejb-name>
      <method-intf>Local</method-intf>
      <method-name>getAccounts</method-name>
      <method-params>
        <method-param>int</method-param>
      </method-params>
    </method>
    <trans-attribute>Never</trans-attribute>
  </container-transaction>
</assembly-descriptor>
```
Choose optimal access intent and isolation level
The access intent service enables developers to tune the management of application persistence. Access intent enables developers to configure applications so that the EJB container and its agents can make performance optimizations for entity bean access.

Seven predefined access intent policies are available. The policies are composed of different attributes. The access type is of primary interest and controls the isolation level, lock type, and duration of locks obtained when bean data is read from the database.

Isolation levels represent how a database maintains data integrity against the problems like dirty reads, phantom reads, and non-repeatable reads, which can occur due to concurrent transactions.

Choosing a right isolation level for your program depends upon your application's requirement. You can reduce the transaction isolation level where appropriate, because the most restrictive and protected access intent policy produces the most performance overhead.

Note: For more details about this configuration, please review these links:

Implement caching options
The EJB container invokes certain methods only once when it creates an EJB and puts it in the pool. You can use these methods as cache, acquiring resources like home EJB references, datasources, or objects, and put them in instance variables. Once you acquire these resources, you do not need to acquire resources for each client, because those resources are already acquired and available.

Remember that you should not acquire physical resources like database connections with these “one time” methods. If you get multiple clients and the pool size is larger, it is better to acquire this type of resources in each method and remove them in that method.

The methods mentioned above depend on the EJB type:
- Session EJB: setSessionContext() or ejbCreate()
- Entity EJB: setEntityContext()
- Message Driven Bean: setMessageDrivenContext() or ejbCreate()
Also, the EJB container calls a method just before removing a bean from the pool. So, whatever resources you acquired in your bean during the creation process must be released in this method. In this case, the methods are:

- Session EJB: ejbRemove()
- Entity EJB: unSetEntityContext()
- Message Driven Bean: ejbRemove()

Example 11-9 shows an example of caching an EJB Home reference in the ejbCreate() method.

**Example 11-9  Caching EJB Home references in session EJB ejbCreate() method**

```java
public void ejbCreate() throws javax.ejb.CreateException, ServiceLocatorException {
    try{
        paciHome    = (PacienteLocalHome)
            KhatuServiceLocator.getInstance().getLocalHome("java:comp/env/ejb/Paciente");
        estuCabHome = (EstudioCabLocalHome)
            KhatuServiceLocator.getInstance().getLocalHome("java:comp/env/ejb/EstudioCab");
        estuDetHome = (EstudioDetaLocalHome)
            KhatuServiceLocator.getInstance().getLocalHome("java:comp/env/ejb/EstudioDeta");
        cartasDetHome = (CartasDetaLocalHome)
            KhatuServiceLocator.getInstance().getLocalHome("java:comp/env/ejb/CartDeta");
    } catch (ServiceLocatorException e) {
        throw e;
    }
}
```

**Remove stateful session beans explicitly**

Instances of stateful session beans have an affinity to specific client requests. They will remain in the container until they are explicitly removed by the client, or removed by the container when they time out.

So, if the client finishes work with the bean and does not remove the bean explicitly, the container keeps the bean in the instance cache or passivates the bean. This process consumes unnecessary memory.
For this reason, it is necessary that you remove the bean explicitly using the remove() method, as shown in Example 11-10.

Example 11-10  Explicitly removing stateful session beans

```java
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.naming.*;
import javax.ejb.*;
import com.ibm.uxo.ejbs.*;

public class BestPracticesServlet extends HttpServlet {

    BestPracticesHome sseHome = null;

    public void init(ServletConfig config) throws ServletException {
        super.init(config);
        try {
            sseHome = EJBhomeCache.getInstance().getMbhHome();
        } catch(Exception e) {
            throw new ServletException("INIT Error: " + e.getMessage(),e);
        }
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

        BestPractices ssmgr = null;

        try {
            ssmgr = sseHome.create(1);
            ssmgr.someBunchOfMethods();
            ssmgr.remove();  // EXPLICITLY REMOVE WHEN DONE!
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

WebSphere EJB Container settings

To check the available settings and best practices for their configuration, please review 7.3.14, “EJB container” on page 189.
Accessing EIS resources

This chapter contains key points which should be considered when a J2EE application with access to one or more EIS is developed. It provides best practices for accessing the following Enterprise Information Systems:

- DB2 for z/OS using JDBC and SQLJ, discussed in 12.2, “Best practices for JDBC and SQLJ” on page 302
- CICS using the J2EE Connector Architecture and the CICS Transaction Gateway, discussed in 12.3, “Best practices for Java 2 Connector Architecture” on page 320
- IMS using the J2EE Connector Architecture and IMS Connect, discussed in 12.3, “Best practices for Java 2 Connector Architecture” on page 320
- WebSphere MQ using the Java Message Service, discussed in 12.4, “Best practices for JMS” on page 330
12.1 General hints

Development environments, such as Rational Application Developer (RAD), WebSphere Application Developer Integration Edition (WSAD IE), or WebSphere Developer for zSeries (WDz), offer the possibility to generate code to access various Enterprise Information Systems (EIS) automatically. It is easier for the developer to use these tools, but they may have a slight impact on performance. In some cases, it may be more efficient to use the available APIs manually. An example of this practice for the Java 2 Connector Architecture (J2C) is using the Common Client Interface (CCI), which is a common API for interacting with resource adapters, and is independent of a specific EIS.

As for J2C resource adapters, this practice is also valid for database access using SQL. Writing your own SQL statements may be more efficient than generating them.

Automatically generated code must be tested to ensure that it can reach the performance goals required. The effort of writing enterprise applications will be much higher if no generated code is used. At this moment, it is hardly possible to optimize automatically generated code without losing the advantages the development environment mentioned above provides.

12.2 Best practices for JDBC and SQLJ

DB2 for z/OS can be accessed from a J2EE application using JDBC or SQLJ. The following sections describe hints and tips to access DB2 in an efficient way.
12.2.1 Data type mapping

For optimal performance, we recommend properly mapping the Java data types used in the application to the DB2 column data types. The main reason for this is to avoid unnecessary conversions and thus reduce performance. Table 12-1 shows a summary of recommended mappings of Java to DB2 data types. Primitive Java types should be used for NOT NULL columns and corresponding wrapper types for nullable columns.

<table>
<thead>
<tr>
<th>Java data type</th>
<th>DB2 data type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>SMALLINT</td>
<td>No direct mapping in DB2; SMALLINT is the best match. Zero indicates false, and any non-zero value indicates true.</td>
</tr>
<tr>
<td>byte</td>
<td>SMALLINT</td>
<td>No direct mapping in DB2; SMALLINT is the best match.</td>
</tr>
<tr>
<td>short</td>
<td>SMALLINT</td>
<td></td>
</tr>
<tr>
<td>int</td>
<td>INTEGER</td>
<td></td>
</tr>
<tr>
<td>long</td>
<td>DECIMAL(19.0)</td>
<td>No 64-bit integer in type in DB2; DECIMAL with precision 19 can hold all long values.</td>
</tr>
<tr>
<td>java.math.BigInteger</td>
<td>DECIMAL(19.0)</td>
<td></td>
</tr>
<tr>
<td>float</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>DOUBLE or FLOAT</td>
<td>FLOAT is a synonym for DOUBLE in DB2.</td>
</tr>
<tr>
<td>java.math.BigDecimal</td>
<td>DECIMAL(p,s)</td>
<td>p = precision, s = scale.</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>CHAR(n)</td>
<td>Fixed-width column of length n.</td>
</tr>
<tr>
<td></td>
<td>VARCHAR(n)</td>
<td>Variable-width column of maximum length n.</td>
</tr>
<tr>
<td></td>
<td>GRAPHIC(n)</td>
<td>Fixed-width column of length n.</td>
</tr>
<tr>
<td></td>
<td>VARGRAPHIC(n)</td>
<td>Variable-width column of maximum length n.</td>
</tr>
</tbody>
</table>
12.2.2 Using static SQL

Static SQL is generally faster than dynamic SQL and should be used wherever it is possible. When using static SQL, parsing and access path calculation is done at compile time and not at runtime. Java’s implementation of static SQL is called SQLJ. Figure 12-1 shows the differences between dynamic and static SQL.

<table>
<thead>
<tr>
<th>Java data type</th>
<th>DB2 data type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte[]</td>
<td>CHAR(n) FOR BIT DATA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARCHAR(n) FOR BIT DATA</td>
<td></td>
</tr>
<tr>
<td>java.sql.Date</td>
<td>DATE</td>
<td></td>
</tr>
<tr>
<td>java.sql.Time</td>
<td>TIME</td>
<td></td>
</tr>
<tr>
<td>java.sql.Timestamp</td>
<td>TIMESTAMP</td>
<td></td>
</tr>
<tr>
<td>java.sql.Blob</td>
<td>BLOB(n)</td>
<td></td>
</tr>
<tr>
<td>java.sql.Clob</td>
<td>CLOB(n)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DBCLOB(n)</td>
<td></td>
</tr>
</tbody>
</table>

![Figure 12-1  Dynamic versus static SQL](image-url)
In some cases, it is not possible to use static SQL, for example, if a GUI application allows a large number of options on your SQL queries. In this case, the dynamic statement caching should be turned on in the DB2 database. You can find more information about this in 10.2.1, “Data source tuning” on page 244.

Example 12-1 and Example 12-2 show the equivalent code necessary in JDBC and SQLJ respectively to select the address for a given name from table EMP.

**Example 12-1  JDBC**

```java
java.sql.PreparedStatement ps = con.prepareStatement("SELECT ADDRESS FROM EMP WHERE NAME=?");
ps.setString(1, name);
java.sql.ResultSet rs = ps.executeQuery();
rs.next();
addr = rs.getString(1);
rs.close();
```

**Example 12-2  SQLJ**

```sql
#sql [con] { SELECT ADDRESS INTO :addr FROM EMP WHERE NAME=:name };
```
The chart in Figure 12-2 shows the results of a simple SQL performance comparison between JDBC and SQLJ. Using JDBC, the SQL statement was preloaded into the dynamic statement cache and always found there during the measurement. The measured SQL statements are:

- Open - 4 Fetch - Close, selecting four rows containing columns of different data types
- Four inserts, inserting four rows containing columns of different data types
- Four singleton selects, selecting four rows containing columns of different data types

Figure 12-2  Comparison JDBC and SQLJ

SQLJ applications are portable even to DBMSs that do not support SQLJ. When the preparations, such as precompile and BIND, for an SQLJ program have not been done, the SQLJ runtime will automatically emit JDBC calls. SQLJ is key to high performance, but does not lock a customer into a particular DBMS. The only performance advantage of using JDBC over SQLJ is where the SQL statements are very complex and literals are provided.
12.2.3 AutoCommit

By default, opening a database connection through the DriverManager class is done with the AutoCommit property set to true. This forces a commit after each single SQL statement, which adds noticeable execution cost, because of:

- Releasing and re-allocating locks
- Unregistering and registering with RRS
- Maintaining JDBC objects

The property should be set to false and a commit should be done only when it is necessary (see Example 12-3). This is not necessary when read-only SQL statements are executed.

Example 12-3 AutoCommit

```java
conn.setAutoCommit(false);
```

Disabling the AutoCommit means that developers will have to prove each time a commit is required or not for every single SQL statement that will be executed in an application. In case a commit is needed, the developer has to force a commit when executing the SQL statement.

12.2.4 Select and update

For optimal performance, we have always recommended that DB2 applications should only select and update columns actually required by the application. This recommendation is even stronger for Java applications, because column processing is one of the major factors in CPU resource consumption. The two primary reasons for this are:

- Strings for character string columns must be converted between Unicode (Java) and EBCDIC/ASCII (DB2 engine).
- A Java object is created per column per row for those data types that are not primitive types in Java, such as character string columns.

Additionally, DB2 should not be used as an expensive file access method like a single row select. The relational functionality (for example, join) can be exploited. WHERE predicates can be used to let DB2 do some filtering. JOIN should be rather used than SUBSELECT.
12.2.5 Numbers

Numbers should be stored into the database using DB2 numeric data types. This saves the overhead of creating an object and of EBCDIC to Unicode conversion because most of the character data is stored in the database using ASCII or EBCDIC encoding schema.

12.2.6 Use DB2 built-in functions

DB2 comes with many useful built-in functions that are more efficient than their Java counterparts. For example, when retrieving a fixed-width character data column, you may want to get rid of the trailing blanks that DB2 appends whenever the value is shorter than the column’s length. You can use the Java String.trim() method shown in Example 12-4.

Example 12-4  Java trim method

```sql
#sql { SELECT JOB INTO :job FROM DSN8710.EMP ... };
job = job.trim();
```

However, it is more efficient to use the DB2 TRIM function shown in Example 12-5, because no intermediate String object has to be created.

Example 12-5  DB2 built-in trim function

```sql
#sql { SELECT TRIM(JOB) INTO :job FROM DSN8710.EMP ... };
```

12.2.7 Releasing resources

It is very important to close and release resources when they are not used any longer. For example, the prepState.close() should be executed before the object prepState is used to prepare another statement. The JDBC Driver maintains its own links to resources. They are only released when the resources are closed or the connection is closed. The Garbage Collector cannot reclaim those objects until they are released, and eventually the application will run out of JDBC resources or memory.

- Close ResultSets when the application is done with them.

  The Garbage Collector cannot reclaim those objects until they are released, and eventually the application will run out of JDBC resources or memory.
Close PreparedStatements that have ResultSets as soon as they are done being used. Closing the ResultSet is not enough to release the underlying cursor resource. If the PreparedStatement is not closed, the cursor resource is tied up for the life of the PreparedStatement.

Close CallableStatements when you are finished with them. The application could run out of call sections.

Resources should be released even in the case of failure. The Java try / finally construct is well suited to achieve this (see Example 12-6).

**Example 12-6  Try / finally construct to ensure proper closing of iterator**

```java
void foo() throws SQLException {
    MyIterator iter = null;
    try {
        #sql iter = { SELECT ... FROM ... };
        while (true) {
            #sql { FETCH :iter INTO ... };
            if (iter.endFetch()) break;
        }
    } finally {
        if (iter != null) iter.close();
    }
}
```

The SQLJ translator automatically generates the code to release statements. With JDBC, the developer must programmatically code the release. The iterators still must be closed programmatically.

### 12.2.8 Connection pooling

The JDBC DataSource connection pooling support reduces the cost of connections to DB2 dramatically. The cost reduction is in two areas:

- **Reusing the DB2 connection thread**
  - A sign on is driven to identify the new user. This also forces writing accounting records.
Keeping the JDBC connection object

The JDBC connection pooling is enabled in the WebSphere Application Server Administrative Console. If the connection pool is not set up there, it will not be possible to use it from the application side. How the connection pool should be set up to improve performance is described in 10.1.1, “Common Connection pool properties” on page 240.

Example 12-7 shows how to define a DataSource. Normally this is only done once by the database administrator.

Example 12-7  DataSource definition

ds = new com.ibm.db2.jcc.DB2DataSource();
ds.setDatabaseName("TESTDB");

Example 12-8 shows how to use a pooled connection within an application.

Example 12-8  Connection pooling

//get connection from pool
Connection Conn1 = ds.getConnection("user","password");
// Turn off auto commit default
Conn1.setAutoCommit(false);
....
Conn1.close();
12.2.9 getXXX() method

The JDBC API defines that each getxxx() method returns a matching Java object (for example, getString() returns a String object). The processing cost of each getxxx() method is mainly determined by the cost of the object’s constructor call. Returning values of Java native data types, like an integer, is much cheaper than returning complex objects, like a Timestamp object. Based on this information, the database can be designed for high performance. You should avoid retrieving “expensive” data types if you do not need them. Just performing a “SELECT *” from a table and later sorting out in the application which fields you will use is too much luxury for a Java application. You should only select the fields you really use in your query. Also, never make it a default in your application to include TIMESTAMP and DATESTAMP in your queries; again, only select them if you really need them. Just one quick look at Figure 12-3 will show you why!

Figure 12-3  Relative cost of getxxx() processing
Figure 12-3 on page 311 shows the relative cost of all $\text{getxxx()}$ methods. Retrieving a Date column is about 21 times more expensive than retrieving a short column.

The JDBC API allows you to use different $\text{getxxx()}$ methods to retrieve a database column, and using a non-matching $\text{getxxx()}$ method is syntactically correct, but it causes a performance overhead per column. The overhead heavily depends on the DB2 data type.

![Figure 12-4: getString() compared to matching $\text{getxxx()}$ method](image)

Figure 12-4 shows the overhead retrieving a column of a certain data type using the $\text{getString()}$ method instead of the matching $\text{getxxx()}$ method in percents. The overhead you will get when a $\text{getString()}$ method is used to receive a SMALLINT instead of the matching $\text{getShort()}$ is about 380%!

### 12.2.10 Use of transactions

Using RRS transactions provides a performance improvement. The following considerations refer to the DB2 Universal JDBC Driver in DB2 UDB for z/OS Version 8. This driver provides both Type 2 and Type 4 support. The type 4 driver support uses a communication protocol to communicate requests from a z/OS Java application to a remote DB2 database. It uses J2EE XA transaction processing to process global transactions. In contrast, the type 2 driver support uses a local protocol to communicate requests from a z/OS Java application to a
target DB2 running on the same z/OS system image as the application. When the Type 2 driver is used on z/OS, the driver supports the use of z/OS Resource Recovery Services (RRS) to coordinate global transactions across multiple resource manages using 2-phase commit processing. The type 2 driver handles all global transactions as RRS-coordinated global transactions. Please see also 12.3.3, “Use of transactions” on page 323.

Also refer to 10.2, “Java Database Connectivity (JDBC)” on page 243 for additional settings and parameters in WebSphere Application Server.

### 12.2.11 Statement caching

In WebSphere Application Server, performance can be increased using statement caching when executable statements are used repeatedly. When a statement is cached before it is re-executed, the code does not have to be reparsed. The statement object does not have to be recreated and parameter size definitions do not have to be recalculated. By default, the statement caching is enabled with a cache size of 5 statements.

The statement cache is enabled and set in the WebSphere Application Server Administrative Console for the kind of Java2 application described in this thesis. More information about the statement cache can be found in 10.2.1, “Data source tuning” on page 244.

The setDefaultStmtCacheSize(int) method is used for the default connection context. For any other connection context, the setStmtCacheSize(int) method is used. For more information about the connection context, see “Explicit connection context objects” on page 317.

### 12.2.12 Update batching

UPDATE, DELETE, and INSERT statements are collected and sent to the database for execution all at once. This saves round trips to the database. Batch statements must be batchable and compatible. Update batching is typically used for an operation that is executed several times, for example, within a loop.

Batchable statements are UPDATE, INSERT, and DELETE when they do not have stream host expressions. In future versions of SQLJ, stored procedure calls and Data Definition Language (DDL) statements may become batchable. Compatible are multiple instances of the same statement. In future versions of SQLJ, instances of different statements may become compatible, if they do not have any host expressions. Update batching is disabled by default.
Example 12-9 shows how to enable batch updating.

**Example 12-9  Enabling update batching**

```java
ExecutionContext ec = new ExecutionContext();
ec.setBatchingEnabled(true);
```

### 12.2.13 Row prefetching

Normally, when JDBC receives a query result, it receives one row at a time. Every single row requires a round trip to the database. Row prefetching enables the reception in groups of multiple rows. The number of rows to be prefetched can be set with the setFetchSize() method of an ExecutionContext instance. Example 12-10 shows how to set a prefetch size to 20 by getting the default execution context of the default connection context.

**Example 12-10  Setting the fetch size with default connection context**

```java
DefaultContext.getDefaultConetxt().getExecutionContext().
    .setFetchSize(20);
```

Example 12-11 shows how to set the fetch size for queries using a given context. In this case, use the underlying JDBC connection and cast it to a Connection instance.

**Example 12-11  Setting fetch size using a given connection context**

```java
((Connection)DefaultConetxt.getDeafultContext().getConnection())
    .setDefaultRowPrefetch(20);
```

There is no maximum row-prefetch value. In JDBC and SQLJ, the default value is 10, which should be fine in most cases. In cases where an application receives a large number of rows, it might be good to increase it.
12.2.14 Summary

The chart in Figure 12-5 shows the principal results of different performance improvements. The chart shows the results of the following tests:

- The first measurement shows a baseline.
- The second measurement uses another WebSphere and JDBC driver, but also a pooled connection instead of creating a new connection. Obviously, in this test, the gain can be found in the portion of getting the connection.
- In the third measurement, dynamic statement caching was enabled and the SQL statement was found in the cache. Obviously, the benefit here is in the portion of preparing the statement.
- The fourth measurement uses SQLJ instead of JDBC to access DB2. When using SQLJ, the portion of preparing the statement disappears completely.

Attention: If you perform the tests for your own specific application, you may get different results, as the relative portions spent on each part (getting a connection, preparing a statement, and so on) may be different than in this specific case. Also note that the results of such a test also depend on many other settings in your environment. We discussed a number of those in this book.

Figure 12-5 Principal performance improvements using JDBC and SQLJ
12.2.15 Additional best practices for SQLJ

The following sections inform you about the possibilities to improve the performance of a J2EE application accessing a DB2 database using SQLJ.

Use positioned iterators, not named iterators

Iterators are the SQLJ equivalent to JDBC result sets. There are two ways to define iterators:

- By column name = named iterator
- By position in the select statement = positioned iterator

Named iterators are implemented as a wrapper around positioned iterators, with an associated hash table that maps column names to column numbers. Named iterators are more convenient to use, but more expensive than positioned iterators.

Example 12-12 and Example 12-13 on page 317 show an example of the usage of a named and a positioned iterator respectively.

For best performance, the use of positioned iterators is recommended, as they do not have as much overhead.

Example 12-12  Named iterator

```sql
// Named Iterator
#sql iterator TestCase001A (short Fkeycr, Time Ftime, BigDecimal Fnum);
....
short wfkeycr;
Time wftime;
BigDecimal wfnum;
...
#sql [myconn] cursor002 = {SELECT FKEY, FTIME, FNUM FROM WRKTB01};
while (cursor002.next()) {
    wfkeycr = cursor002.Fkeycr();
    wftime = cursor002.Ftime();
    wfnum = cursor002.Fnum();
}
```
Example 12-13  Positioned iterator

// Positioned Iterator
#sql iterator TestCase001(short, Time, BigDecimal);
....
short wfkeycr;
Time wftime;
BigDecimal wfnum;
...
#sql [myconn] cursor001 = {SELECT FKEY, FTIME, FNUM FROM WRKTB01};
#sql {FETCH :cursor001 INTO :wfkeycr, :wftime, :wfnum};

Always customize with online checking enabled
We strongly recommend customizing the SQLJ profile using the online checker. The online checker is called by:
db2profc ... -online=<db2_location_name>

The online checker accesses the DB2 catalog to check JDBC/SQLJ-supported compatibility and convertibility processing and to determine the length of string columns. Java String objects do not have a concept of length, and can only be obtained from the catalog. In order to have the predicates considered for index access, the information in the Database Request Module (DBRM) must match definition in the DB2 catalog by data type and length. Online checking adds that information to the DBRM.

Not only are CHARACTER columns affected, but so are numeric columns. The optimizer will choose a non-matching index scan when the use of a host variable of type LONG to match a column of type INTEGER.

The serialized profile should be recustomized after each run of the SQLJ translator and available via the CLASSPATH at runtime.

If the SQLJ serialized profile is not customized, the Java application will execute dynamically using JDBC.

Explicit connection context objects
The default connection context object for a program is stored in a static variable of the default connection context class. In a multi-context environment (like WebSphere Application Server), the use of a default connection context is not thread-safe and must not be used.
An additional risk is the usage of the default connection context, which implicates a throughput bottleneck. Closing the context releases the resources maintained by the connection context (like statement handles) and closes the underlying database connection. When the constant KEEP_CONNECTION is passed as an argument, the context will be closed, but the database connection will be retained. This avoids new effort to get the connection when it is needed again.

Example 12-14 gives the explicit connection context.

```
// Connection context declaration
#sql context ctx;
...
//get context
myconn=new ctx(Conn1);
...
//use context in SQL
#sql [myconn] {set transaction isolation level read committed};
...
#sql [myconn] cursor001 = {SELECT FKEY,FSMALLINT,FINT FROM WRKTB01 WHERE FKEY >= :wfkey};
...
//close context but keep database connection
myconn.close(ConnectionContext.KEEP_CONNECTION);
```
Figure 12-6 shows a simple performance measurement. With an explicit context, the number of transactions per second is nearly two times higher than with a default context.

**Check explain tables**

Whenever DB2 data is read or written, an SQL statement is executed. Once DB2 accepts a statement, an execution plan is created. The execution plan defines how the DBMS finds and writes data. For example, the DBMS has to decide whether an index is used or not and which index is used. To discover this information, the execution plan can be read out. Therefore the SQL statement EXPLAIN PLAN is used.

```sql
EXPLAIN PLAN [SET STATEMENT_ID=<id>; INTO <table>] FOR <SQL-Statement>
```

When EXPLAIN PLAN is executed, DB2 writes the results into the PLAN_TABLE.

It is a good idea to always bind with EXPLAIN(YES) and to check the PLAN_TABLE for potential performance problems, for example, table space scan, merge scan join, and non-matching index scan.

See Chapter 26, “Using EXPLAIN to improve SQL performance”, of the Application Programming and SQL Guide, SC26-9933, for information about how to set up and interpret a PLAN_TABLE.
Alternatively or additionally, you can use DB2 Visual Explain Version 8, which is a free feature of DB2 for z/OS. It lets you graphically analyze the access paths that DB2 chooses, which eliminates the need to manually interpret the plan_table output. You can download Visual Explain at:


Rebind packages regularly
It is not recommended to rebind static plans and packages after each and every REORG. Instead, statistic data, RUNSTATS, should be collected with a REORG. The recommendation is to rebind all plans/packages at least once within the life of any given release and when the RUNSTATS have changed significantly, that is, over 10%.

12.3 Best practices for Java 2 Connector Architecture

The paragraphs below give provide practices on how the J2EE Connector Architecture (JCA or J2C) should be used to access Enterprise Information Systems like CICS and IMS.

12.3.1 Re-use of objects
Caching the connection factory is probably the biggest single performance enhancement that can be made, because it avoids JNDI lookups and reduces the I/O. The connection factory can be cached on the call:

(ConnectionFactory)ic.lookup(fqndiName)

into a static variable. This gives a significant path length reduction by avoiding the lookup to JNDI each time a connection is required. The main benefit is in the CPU utilization reduction. Caching the initial context javax.naming.Context into a static variable provides also some path length reduction, which improves performance.
Figure 12-7 shows the path length in milliseconds of CPU per CICS transaction when re-using the initial context (IC), the connection factory (CF), both and neither with different communication area (COMMAREA) sizes.
Figure 12-8 shows the throughput as the CICS transaction rate, that is, transactions per second, when re-using the initial context, the connection factory, and both and neither with different COMMAREA sizes.

Both graphs are based on a measurement of a two-phase commit using the local mode of CICS Transaction Gateway.

The JNDI lookup as well as obtaining the connection should not be done within the ejbCreate() method. The usage of unspecified transaction contexts is one of the biggest holes in the EJB specification and assures non-portability.
12.3.2 Managed environment

In a managed environment, the application components and the resource adapter are connected with the application server through contracts. This means that the application server manages the connection pooling, transactionality, and security for the application. In contrast to a non-managed environment, the implementation of the ConnectionManager interface happens within the application server. The access on the resource adapter is regulated by the application server via system contracts.

The use of the managed environment is a key benefit because it is one way to exploit XA and RRS transactions for two-phase commit and JCA connection pooling functions in WebSphere. The exploitation is achieved by defining a connection factory within the WebSphere Application Server Administrative Console or through WebSphere Admin (wsadmin) scripting. The connection factory can be looked up through a resource reference within the JCA code:

cfA = (ConnectionFactory) ic.lookup(cfRefA);

12.3.3 Use of transactions

A big advantage of using the managed environment is that all transaction management can be delegated to the Web/EJB containers.

Resource Recovery Services (RRS) transactions provide better performance than XA transactions. This is because the resource adapters using RRS transactions normally use native z/OS calls and resource-specific interfaces instead of TCP/IP connectivity used with XA resource managers. They avoid the overhead that comes with XA transactions.

Resource Recovery Services (RRS) is used when connecting locally (not using TCP/IP) between WebSphere Application Server on z/OS and:

- IMS using the IMS Connector for Java and IMS Connect
- IMS using the IMS JDBC connector
- CICS using CICS Transaction Gateway
- DB2 for z/OS using a Type 2 driver
- WebSphere MQ using bindings mode

All RRS compliant resource adapters are required to support the property RRSTransactional in their ManagedConnectionFactory and must support a getter method for the property.
Example 12-15 shows an example of using RSS transactions.

Example 12-15   Using RRS transactions

```java
java.lang.Boolean RRSTransactional=true;

java.lang.Boolean getRRSTransactional(){
    // Determine if the adapter can run RRSTransactional based
    // on it's configuration, and set the RRSTransactional property
    // appropriately to true or false.
    return RRSTransactional;
}
```

RRS support is only applicable in a "local" environment, where the back end must reside on the same system image. CICS and IMS resources adapters may use RRSTransactional support only when these adapters are configured to use local interfaces to their back-end resource manager, which, as stated above, must reside on the same system image as the IBM WebSphere Application Server for z/OS.

These adapters are also capable of being configured to a remote instance of their back-end resource manager. In this case, the adapters will respond "false" when the getRRSTransactional() method is invoked and instead of running as RRSTransactional, they will use whichever one of the three types of J2EE Transaction support they have chosen to support. With the CICS TG in local mode, the RRS transactional mode is automatically exploited.

Another point to look at to improve performance is the commit mode during a transaction. It should be proved if it is necessary to run your commit modes with SyncLevel Confirm for a one-phase commit or SyncPoint for a two-phase commit. This will not be necessary for read-only transactions. Using SyncLevel Confirm or SyncPoint blocks the IMS until your WebSphere Application Server application commits or until WebSphere Application Server has finished waiting for all other units of work involved.

More detailed information about the usage of RRS in WebSphere connectivity can be found in Chapter 8, "WebSphere Application Server for z/OS", of Systems Programmer's Guide to Resource Recovery Services (RRS), SG24-6980.
12.3.4 Connection pooling

*Connection pooling* is maybe the key benefit of the JCA managed environment. All actual socket connections can be managed by the WebSphere Application Server Pool Manager and configurable limits can be set and monitored within WebSphere Application Server. Therefore, the application developer only needs to obtain a connection handle and the underlying managed connection is handled by the JCA infrastructure. The connection is acquired using the following statement:

```java
eciConn = (Connection) cf.getConnection(ecf);
```

The first use of each connection in the pool may take longer as the physical socket is established. To prevent this, you can write a simple application that primes the pool to establish each connection. With the CICS TG implementation, the `getConnection()` method does no I/O and this does not occur until the first usage of the connection by an interaction execute. The behavior of a specific implementation apart from CICS TG has to be determined.

Please refer to 10.1.1, “Common Connection pool properties” on page 240.

12.3.5 Connection usage

When using the J2EE Connector Architecture (JCA) Version 1.0 in WebSphere Application Server Version 5.x, the connection usage should follow the “get-use-close” model. This means that an application always obtains a new connection when it needs one, then uses it and then closes it again when the work is done. This might sound inefficient, but the connection pooling the application server implements makes the `get()` operation cheap. Also, different instances or parts of the application can reuse the connection because the application holds on to the connection for only as long as it is needed. Therefore, the total resource usage is reduced.

When using the J2EE Connector Architecture (JCA) Version 1.5 in WebSphere Application Server Version 6.x, the connection usage should follow the “cached-handle” model. This means that an application obtains the connection once up-front and caches a reference to it in an instance field. This allows the programmer to delay the close() method on a connection. Instead, the J2C infrastructure will disassociate the managed connection from the connection handle when the transaction scope ends. This allows the JCA infrastructure to efficiently handle lazy use of connections. Furthermore, it frees up the developer from the concern of worrying about connection handles.

IBM WebSphere Application Server Version 5.x addressed the drawbacks of the cached-handle connection model with an extension to the JCA 1.0 specification known as “smart connection handles”.
With the CICS TG implementation, the advantages of connection pooling are only evident when a remote gateway is used, because the local mode of the CICS TG does not use any I/O for access.

### 12.3.6 Lazy association

Rather than re-associating the connection handle with the managed connection the next time a method is called, the optimization uses lazy association. If the method does not use the connection, or it only calls simple methods on the connection handle that do not require access to the back end, a managed connection is not removed from the pool unnecessarily. Instead, when the connection handle determines that it does need to be reassociated to a managed connection, it can cast the connection manager to a LazyAssociatableConnectionManager and call the associateConnection method.

This method takes the connection handle as the first parameter, followed by the managed-connection factory, and requests information passed on the initial call to allocateConnection. The connection manager then finds another suitable managed connection from the pool and uses the managed connection's associateConnection method to tie it to the connection handle.

#### Example 12-16 Interfaces for dissociation and lazy association

```java
public interface DissociatableManagedConnection {
    void dissociateConnections() throws ResourceException;
}

public interface LazyAssociatableConnectionManager {
    void associateConnection(Object connection,
        ManagedConnectionFactory mcf,
        ConnectionRequestInfo cxReqInfo)
    throws ResourceException;
}
```

This allows the application developer to forget about closing down connections.
12.3.7 Lazy enlistment

Transactions, particularly XA and RRS (global) transactions, are expensive. This makes it more important for a transaction not to do more work than necessary. Interfaces in JCA 1.5 prevent unnecessary enlistment of XAResource objects. You should not enlist in the transaction unless absolutely necessary. The solution therefore is a lazy enlistment.

Example 12-17 Interfaces for lazy enlistment

```java
public interface LazyEnlistableManagedConnection {
}

public interface LazyEnlistableConnectionManager {
    void lazyEnlist(ManagedConnection mc)
        throws ResourceException;
}
```

The LazyEnlistableManagedConnection interface is a marker interface implemented by the managed connection to indicate to the connection manager that it does not need to eagerly enlist the managed connection in an existing transaction when a new connection is created in a transaction or in a new transaction started when a connection already exists. If a connection handle is about to perform some work that should be part of any transaction, and its managed connection has not already been enlisted, it should determine whether the connection manager implements the LazyEnlistableConnectionManager interface. If it does, it should call the lazyEnlist method passing the managed connection. This method returns nothing, but if a transaction is associated with the calling thread, the XAResource from the managed connection is enlisted at that point. If the connection is not enlisted, it needs to call lazyEnlist again before each subsequent piece of work in order to check that a transaction has not been started since the last time it called the method.

Lazy Connection Enlistment applies when a connection factory is referenced in a resource reference but is not used in a transaction. This means that no interactions are sent to it and so it will not be enlisted.

This kind of enlistment only applies to XA transactions and not to transactions created in a CICS TG local mode.
12.3.8 Best practices for CICS Transaction Gateway

The CICS Transaction Gateway uses the J2EE Connector Architecture to work with CICS and the information provided in the previous sections of 12.3, “Best practices for Java 2 Connector Architecture” on page 320 applies here as well. In addition, there are some additional key points that apply to the usage of the CICS Transaction gateway specifically. We go over these in the following paragraphs.

Transmitting data in a COMMAREA

The CICS TG, in combination with CICS Transaction Server, provides built-in data compression for ECI flows. Any trailing nulls in the COMMAREA payload sent between the JCA resource adapter, the Gateway daemon, and CICS are automatically removed when sent over any of the supported network connections. This compression is dynamic and is not visible to either the J2EE client application or the CICS application. To best exploit the functionality, there are two suggested approaches:

1. If the length of the input and output data structures are known, then we suggest that the J2EE client application builds a record containing just the required input information, and then sets the specified COMMAREA length to the length of the COMMAREA data structure to be used in CICS. This COMMAREA structure should be equal to the size of the data to be returned by CICS. In addition, if the J2EE client application needs to override how much data is returned, the payload should specify this ahead of time using the setReplyLength method to receive a truncated amount of data.

2. If the length of the input or output data structure cannot be determined, then it is suggested that the J2EE client application builds a record containing the required input information, and then sets the specified COMMAREA length to be the maximum possible (that is, 32 KB). The CICS application must then be made capable of handling a full 32 KB of payload and should ensure this data structure is initialized to binary zeroes. The CICS application should then return as much information as necessary, making sure to efficiently utilize the space within the COMMAREA payload, leaving any empty data as trailing binary zeroes. The J2EE client application will receive the returned data, but all trailing nulls will be removed from the data sent across any network connections.
Data conversion
When writing Java applications to invoke CICS programs, data conversion is a key issue, because CICS evolved in an EBCDIC world, while Java is based on Unicode. Normally, you would convert Java Strings that are stored in Unicode within the JVM to an EBCDIC byte array, which is required by CICS. The alternative is to convert the data to ASCII within the JVM and then convert from ASCII to EBCDIC within CICS. Data conversion from Unicode to ASCII is an efficient operation in Java, as it involves only the removal of the high-order byte, while conversion to EBCDIC requires a table lookup. This means that the high cost of EBCDIC conversion can be transferred to CICS, therefore potentially improving performance within the JVM.

In this case you would use an ASCII code page, such as 8859_1, when creating the byte array:

```java
byte abCommarea[] = new byte[27];
abCommarea = "abcd".getBytes("8859_1");
```

After receiving the byte array back from CICS, convert it to a String as follows:

```java
String strCommarea = new String(abCommarea,"8859_1");
```

Please refer to 10.4, “CICS Transaction Gateway” on page 257 for additional information about settings and parameters related to the CICS Transaction Gateway.

12.3.9 IMS Connect
If you want to have a closer look at your IMS connections, there is a tool to do performance analysis for IMS Connect called IMS Connect Extensions for z/OS:


The interpretation from data collected by the IMS Connect Extensions for z/OS can be done with the IMS Performance Analyzer for z/OS:


Please refer to 10.5, “IMS Connect” on page 261 for additional information about settings and parameters related to IMS Connect.
12.4 Best practices for JMS

The Java Message Service (JMS) is used to access WebSphere MQ from J2EE applications. The following sections will give hints and tips to optimize performance when JMS is used.

We will discuss the following areas of concern:

- Connection-related considerations in 12.4.1, “JMS connection considerations” on page 330
- Session-related considerations in 12.4.2, “JMS session considerations” on page 332
- Destination-related considerations in 12.4.3, “JMS destination considerations” on page 333
- Considerations regarding the message producer and consumer in 12.4.4, “JMS message producer / consumer considerations” on page 334
- Message-related considerations in 12.4.5, “JMS message considerations” on page 336

We conclude the JMS part with a few notes on performance testing of JMS applications in 12.4.6, “JMS performance testing” on page 337.

12.4.1 JMS connection considerations

1. Start the connection when appropriate.

   If you start a connection before starting the subscriber/receiver (consumer), then the messages have to wait in the JMS server or they persist if they are durable messages. This may be unnecessary overhead and to avoid this, ensure the consumer is ready to accept messages before starting its connection.

2. Start consumers before producers.

   If you are starting a system, it may help to start the consumers first. This avoids the possibility of excess buffering/persisting of messages during the interval between a producer starting and the corresponding consumer starting. This is most likely to happen with point-to-point or with durable subscriptions.
3. Process messages concurrently

JMS provides a facility to process incoming messages concurrently by using a ConnectionConsumer that pools session objects and handles the distribution of work among them. You can create ConnectionConsumer using the methods in Example 12-18 and Example 12-19.

**Example 12-18  ConnectionConsumer for queues**

```java
public ConnectionConsumer createConnectionConsumer(Queue queue,  
    String messageSelector, ServerSessionPool sessionPool, int  
    maxMessages) throws JMSException
```

**Example 12-19  ConnectionConsumer for topics**

```java
public ConnectionConsumer createConnectionConsumer(Topic topic,  
    String messageSelector, ServerSessionPool sessionPool, int  
    maxMessages) throws JMSException
```

In these methods the main parameters are maxMessages and ServerSessionPool. maxMessages denote the maximum number of messages that can be simultaneously assigned to a server session. ServerSessionPool is an administered object that you configure in vendor specific manner.

4. Close the connection when finished

Always call the close() method on JMS connection and session objects when they are no longer needed. This releases the underlying resource handle. This is especially important for the publish-subscribe model, where clients need to deregister from their subscriptions. Closing the objects allows the queue manager to release the corresponding resources in a timely fashion; failure to do so can affect the capacity of the queue manager for large numbers of applications or threads.
12.4.2 JMS session considerations

A session is used to create multiple producers and consumers. A session can be a QueueSession for a point-to-point or a TopicSession for a publish-subscribe model.

1. Choose proper acknowledgement mode.

When you create a session object, you can choose anyone of the three acknowledgement modes: AUTO_ACKNOWLEDGE, CLIENT_ACKNOWLEDGE, or DUPS_OK_ACKNOWLEDGE.

- CLIENT_ACKNOWLEDGE mode is not a feasible option (when you have the freedom to choose from the other two options), because the JMS server cannot send subsequent messages until it receives an acknowledgement from the client.
- AUTO_ACKNOWLEDGE mode follows the policy of delivering the message once-and-only once, but this incurs an overhead on the server to maintain this policy.
- DUPS_OK_ACKNOWLEDGE mode has a different policy of sending the message more than once, thereby reducing the overhead on the server (imposed when using the AUTO_ACKNOWLEDGE), but imposes an overhead on the network traffic by sending the message more than once. But, the AUTO_ACKNOWLEDGE mode cleans up resources early from the persistent storage/memory, which reduces the overhead because of that.

2. Control transaction.

In the code shown in Example 12-20 and Example 12-21, the first parameter indicates the session is a transactional session. The session objects have commit(), rollback(), and isTransacted() methods to deal with transactional messages.

Example 12-20  Transactional messages for topics

```java
topicSession =
tConnect.createTopicSession(true,Session.AUTO_ACKNOWLEDGE);
```

Example 12-21  Transactional messages for queues

```java
queueSession =
qConnect.createQueueSession(true,Session.AUTO_ACKNOWLEDGE);
```
The problem here is that a transaction starts implicitly when a session is created and ends when a commit() or rollback() method is called. At this stage, after calling a commit() or rollback() method, one more transaction starts implicitly, because there is no explicit method (begin() method) to start a transaction. So, there are a chain of transactions that depend upon commit() or rollback() method calls. Transactional messages are cumulated in the JMS server until the transaction is committed or rolled back. This imposes significant overhead on a JMS server.

3. Close the session when finished.

The reasons for closing sessions are described in 12.4.1, “JMS connection considerations” on page 330.

12.4.3 JMS destination considerations

The destination is a virtual channel between producers and consumers.

Producers send messages to the destination which in turn deliver messages to consumers. The following parameters can be configured:

- Maximum size of the destination
- Maximum number of messages in the destination
- Priority of messages
- Time to live
- Time to deliver
- Delivery mode
- Redelivery delay
- Redelivery limit

You need to perform a JNDI lookup to get the destination object.

For non-durable messages, the time that messages take to deliver to the destination depends upon its number and destination size. If a large number of messages collect in a destination, they take more time to deliver. Set a smaller destination size and smaller number of maximum messages to the destination to improve performance.

Redelivery delay time defines when to redeliver a message if a failure occurs. If this is less, the frequency of redelivery of a message is high, thus increasing network traffic and vice versa. So, high redelivery delay time gives better performance. Redelivery limit defines the number of times a message should be redelivered. Although the probability of guaranteed messaging is less, if the Redelivery limit is less, then the performance is better because the memory overhead for non durable messages and persistent overhead for durable messages is reduced.
12.4.4 JMS message producer / consumer considerations

The producer sends messages to the destination where a consumer consumes messages from the destination. The message producer/consumer is created by the session object.

You send the messages using the producer. Example 12-22 and Example 12-23 provide two samples of sending a message using a topic and a queue respectively.

**Example 12-22  Publishing messages to a topic**

```java
publisher.publish(Message message);
// or
publisher.publish(Topic topic, Message message, int deliveryMode,
                  int priority, long timeToLive);
```

**Example 12-23  Sending messages to a queue**

```java
sender.send(Message message);
// or
sender.send(Queue queue, Message message, int deliveryMode,
            int priority, long timeToLive);
```

The parameters DeliveryMode and TimeToLive are important from a performance perspective. You can provide values for these parameters when you configure ConnectionFactory or destination or when you send a message.

1. Choose non-persistent messages where appropriate.

   Delivery mode defines whether the message can be persistent or non-persistent. This parameter ensures that message delivery is guaranteed.

   For persistent messages, the delivery mode value is `Deliverymode.PERSISTENT`, and for non-persistent messages, the delivery mode value is `Deliverymode.NON_PERSISTENT`. 
If you define the delivery mode as persistent, then the message is stored by the JMS server before delivering it to the consumer, as shown in Figure 12-9.

![Persistent message flow](image)

Figure 12-9  Persistent message flow

If you define the delivery mode as non-persistent, then the message is delivered to the consumer without being saved by the JMS server, as shown in Figure 12-10.

![Non-persistent message flow](image)

Figure 12-10  Non-persistent message flow

The above figures clearly show the difference between the two delivery modes. When using the persistent delivery mode, each message has to be stored by the JMS server either in the database or the file system before delivery of message to consumer and removed after delivery of message. So as far as possible, restrict the use of persistent delivery mode unless and until absolutely necessary.

2. Set Time to live value properly.

You can set the age of the message by setting the Time to live parameter after which the message expires. By default, the message never expires, so you should set the optimal message age so as to reduce memory overhead.
3. Receive messages asynchronously.

You can receive messages synchronously or asynchronously. To receive asynchronous messages, you need to implement the MessageListener interface and onMessage() method. For receiving synchronous messages, you need to use any of the following methods of MessageConsumer:

- receive();
- receive(long timeout);
- receiveNoWait();

The first method blocks the call until it receives the next message, the second method blocks until a timeout occurs, and the last method never blocks.

Normally, when using asynchronous messaging, the calls are never blocked, so a better option to improve performance is to receive messages asynchronously by implementing MessageListener.

The implementation for WebSphere MQ actually does a receive(5000) in a loop under the covers so it is not true asynchronous and may actually increase the number of MQ operations if a larger number would have been chosen.


When not needed anymore the message producers and consumers should be closed.

12.4.5 JMS message considerations

A Message object contains information that is passed between applications. It contains information as payload, headers, and properties. As per performance perspective, you need to mainly consider the type of message, whether it is a Text, Object, Byte, Stream, or Map message. The higher level message types like Map, Stream and Object need more processing to convert them to and from the internal transmission format. These types should not be used unless they provide exactly the functionality the application requires. ObjectMessage carries a serialized Java object and when you choose ObjectMessage, you need to use "transient" keyword for variables that need to be not sent over the network to reduce overhead. Also, ByteMessage takes less processing in the JMS client than a TextMessage, because it is the simplest format.
12.4.6 JMS performance testing

If you want to have a closer look at your JMS connections, there is a tool to do performance analysis for JMS called IBM Performance Harness for Java Message Service:

http://alphaworks.ibm.com/tech/perfharness

Please also refer to 10.3, “Java Message Service (JMS) and WebSphere MQ” on page 250 for additional settings and parameters.
Appendixes
Installation of the Trade V6.1 application

During our project, we used the Trade V6.1 application and we found it useful to create a description of the setup of Trade V6.1.
Overview of the Trade V6.1 application

*Trade* is a WebSphere end-to-end benchmark and performance sample application and is modeled after an online stock brokerage. Trade leverages J2EE components, such as servlets, JSP files, enterprise Beans, Message-Driven Beans (MDBs) and Java Database Connectivity (JDBC), to provide a set of user services, such as login and logout, stock quotes, buy, sell, and account details through standards-based HTTP and Web services protocols.

Trade utilizes the WebSphere Service Integration Bus (SI Bus) for internal message queuing, including publish and subscribe, using Java Messaging Service (JMS). Data from the SI Bus is stored on a local Derby database.

**Note:** Cloudscape™ is no longer shipped with WebSphere. The open source database Apache Derby has taken its place. More information can be found on the Apache Derby Web site:

http://db.apache.org/derby/

The version used for our project was V6.1, which you can download (after signing in or registering with your IBM ID) at:

Figure A-1 shows the overall topology of the Trade V6.1 application. The Trade application can be used with multiple types of clients that utilize multiple protocols.

![Trade V6.1 topology](image)

**Figure A-1  Trade V6.1 topology**

**Derby setup**

We used Apache Derby (Derby) in our setup for the data store that would be used by the SI Bus.

**Generating the data definition**

WebSphere comes with a small script that will generate the Data Definition Language (DDL) for the SI Bus. This DDL will be used by Derby to create the tables necessary.

Generate the DDL as follows:

```
<WAS_HOME>/bin/sibDDLGenerator.sh -system derby -platform zos -create -statementend ";" > <TMP_DIR>/IBMWSSIB.ddl
```

Creating the Derby database

Derby is installed in the WebSphere Deployment Manager directory.

1. Change to the Derby directory:
   ```bash
cd <WAS_HOME>/derby/bin/embedded
   ```
2. Launch the Derby command interface:
   ```bash
   ./ij.sh
   ```
3. Create a new database called *PFClusterBusDB*:
   ```bash
   connect
   ‘jdbc:derby:<WAS_HOME>/derby/databases/PFClusterBusDB;create=true’;
   ```
4. Create the SI Bus tables:
   ```bash
   run ‘<TMP_DIR>/IBMWSSIB.ddl’;
   ```
5. Quit the command interface:
   ```bash
   quit;
   ```

DB2 setup

We used DB2 for our datastore for the Trade V6.1 data portion. The database and accompanying tables are created with the supplied Job Control Language script (JCL). This exists under the z/OS directory of the Trade main directory and is called trade6db_package.jcl. This file must be sent to z/OS using the File Transfer Protocol (FTP) and modified to correspond to system standards. Submitting JCL will create the necessary database and tables.

WebSphere cluster configuration

In our environment, we have the following configuration:

- One cell: pfcell
- Three nodes: pfdmnode (Deployment Manager), pfnodea, and pfnodeb
- One cluster: PFITSO_CLUSTER
- Two cluster members: pfsr01a on pfnodea, and pfsr02b on pfnodeb
Figure A-2 shows the cluster configuration we used in our tests. We had one cluster defined with two servers on two separate nodes in the same cell.

![Server clusters](image)

**Figure A-2  Cluster members**

**J2C authentication**

In our tests, we had administrative security and application security enabled. The Trade application needs to access the database and the local operating system. For these cases, J2C authentication must be set up.

1. Create a new J2C authentication.

   Select **Security → Secure administration, applications, and infrastructure → Java Authentication and Authorization Service → J2C authentication data → New.**
2. Add the definition for the database, as shown in Figure A-3. The Alias field is set to TradeDataSourceAuthData, the User ID is set to the owner of the database, and Password is the password for the user.

3. Click OK.

4. Add a new J2C authentication for the local operating system by clicking New.
5. Add the definition for the user, as shown in Figure A-4. The Alias field is set to **TradeOSUserIDAuthData**, the User ID is set to the user that can access the local operating system, and **Password** is the password for the user.

![Figure A-4  J2C authentication for the local operating system](image)

6. Click **OK**.

**Note**: WebSphere V6.1 has introduced a new user, called WSOWNER. This user is needed to create some aspects of the WebSphere environment and should have access to the local operating system.

**SI Bus setup**

The SI Bus is used by the Trade application to buy and sell stocks. Information is passed back and forth using internal messaging, which uses the Java Message Service (JMS) protocol.

1. Create a new SI Bus.

   Select **Service integration** → **Buses** → **New**.
2. Add the definition, as shown in Figure A-5. The name of the bus is PFITSO_CLUSTER_BUS and the **Bus security** is enabled.

![Figure A-5 Adding a new SI Bus](image)

3. Click **Next**.

4. Click **Finish**, as shown in Figure A-6.

![Figure A-6 Confirmation of the new SI Bus](image)

5. The SI Bus allowed transports must be specified.

   Select **Service integration → Buses → PFITSO_CLUSTER_BUS → Additional Properties → Security.**
6. SI Bus security should be configured, as in Figure A-7. The **Enable bus security** is checked, the Inter-engine authentication alias is set to the **TradeOSUserIDAuthData**, and Permitted transports is set to Allow the use of all defined transport channel chains.

   ![Figure A-7   SI Bus security](image)

7. Click **OK**.

8. The members of the SI Bus are must be specified.

   Select **Service integration** → **Buses** → **PFITSO_CLUSTER_BUS** → **Topology** → **Bus members** → **New**.
9. The Bus member should be configured, as in Figure A-8. The Cluster radio button is selected and the name of the cluster, **PFITSO_CLUSTER**, is selected in the drop-down list.

![Figure A-8  Adding the cluster to the Bus](image)

10. Configure the message store, as shown in Figure A-9. Select the Data store radio button, as Derby has been set up as the data store.

![Figure A-9  Selecting the message store for the Bus](image)
11. Configure the properties for the data store, as shown in Figure A-10. The Data source JNDI name is set to jdbc/pfitso_cluster_bus_ds, the Schema name is set to IBMWSSIB, and the Create tables check box is checked.

![Figure A-10  Datastore properties for the Bus](image)

12. Confirm the creation of the addition of the bus members by clicking Finish, as shown in Figure A-11.

![Figure A-11  Confirmation window for adding new members to a Bus](image)

13. Create queue destinations for the Bus.

   Select Service integration → Buses → PFITSO_CLUSTER_BUS → Destination resources → Destinations → New.
14. Create a new queue destination, as shown in Figure A-12. Select the **Queue** radio button and click **Next**.

![Figure A-12  Create a queue destination](image)

15. Add the queue name TradeBrokerJSD, as shown in Figure A-13.

![Figure A-13  Define the queue name for the destination](image)

16. Assign the queue to the Cluster by selecting **Cluster=PFITSO_CLUSTER** in the drop-down box, as shown in Figure A-14.

![Figure A-14  Select the Bus Member for the queue](image)
17. Click **Finish** to create the new queue, as shown in Figure A-15.

![Image of the Create new queue dialog](image1)

*Figure A-15  Confirmation for adding the new queue to the Bus*

18. Create topic destinations for the Bus.

   Select **Service integration** → **Buses** → **PFITSO_CLUSTER_BUS** → **Destination resources** → **Destinations** → **New**.

19. Create a new topic space, as shown in Figure A-16. Select the **Topic space** radio button and click **Next**.

![Image of the Create new destination dialog](image2)

*Figure A-16  Selecting a new topic space for the Bus*
20. Add the topic space name Trade.Topic.Space, as shown in Figure A-17.

![Create new topic space](image1)

Figure A-17  Defining the topic space name for the Bus

21. Click **Finish** to add the topic space to the Bus, as shown in Figure A-18.

![Create new topic space](image2)

Figure A-18  Confirmation of the new topic space

22. The correct security must be added to the SI Bus for the Application Servers to utilize the bus. Select **Service Integration** → **Buses** → **PFITSO_CLUSTER_BUS** → **Security for bus PFITSO_CLUSTER_BUS**.

23. Click the **New** button to add a new security specification for the Bus.
24. For the purposes of testing the application, select the **Everyone - Allow unauthenticated users to connect to the bus** radio button, pictured in Figure A-19, which will allow everyone access to the Bus.

25. Click the **OK** button to add the new security role.

![Figure A-19  SI Bus Security roles](image)

### JMS setup

We used WebSphere internal messaging to pass the messages from the topics to queues and to the Message Driven Beans (MDB). The physical queues, connection factories, and activation specifications need to be defined. We defined all attributes at the cluster level.

1. Create a new queue connection factory.

   Select **Resources → JMS → JMS providers**.
2. From the Scope drop-down list, select **Cluster=PFCLUSTER**, as shown in Figure A-20.

![Figure A-20](image)

*Figure A-20  Selecting the scope from the drop-down list shows only the items attributed to that scope*

3. Click **Default messaging provider**.

4. Under Additional properties, select **Queue connection factories**, as shown in Figure A-21.

![Figure A-21](image)

*Figure A-21  Default messaging window*
5. Click **New** to create a new queue connection factory (QCF).

6. In the Administration section in the Name field, add TradeBrokeQCF, and in the JNDI name, add jndi/TradeBrokerQCF, as shown in Figure A-22.
7. In the Connection section, select **PFITSO_CLUSTER_BUS** from the Bus name drop-down list, as shown in Figure A-23.

*Figure A-23  Selecting the connection type for the QCF*
8. In the Advanced configuration section, select **TradeOSUserIDAuthData** from the Component-managed authentication alias drop-down list, as shown in Figure A-24.

![Figure A-24](image)

9. Click **OK** to accept the new QCF.

10. Create a new queue connection factory.

   Select **Resources → JMS → JMS providers**.

11. From the Scope drop-down list, select Cluster=PFCLUSTER.

12. Click **Default messaging provider**.

13. Click **Topic connection factories**.

14. Click **New** to create a new topic connection factory.
15. In the Administration section in the Name field, add TradeStreamertCF, and in the JNDI name, add jms/TradeStreamertCF, as shown in Figure A-25.

![General Properties](image)

*Figure A-25 Administration section for a new TCF*
16. In the Connection section, select **PFITSO_CLUSTER_BUS** from the Bus name drop-down list, as shown in Figure A-26.

![Connection section for a new TCF](image)

*Figure A-26  Connection section for a new TCF*
17. In the Advanced configuration section, select **TradeOSUserIDAuthData** from the Component-managed authentication alias drop-down list, as shown in Figure A-27.

![Figure A-27: The remaining sections of a new TCF](image)

18. Click **OK** to create the new TCF.

19. Create a new queue.

   Select **Resources → JMS → JMS providers**.

20. From the Scope drop-down list, select **Cluster=PFCLUSTER**.

21. Click **Default messaging provider**.
22. Click **Queues**.

23. Click **New** to create a new queue.

24. In the Administration section, add **TradeBrokerQueue** to the Name and **jms/TradeBrokerQueue** to the JNDI name, and in the Connection section, select **PFITSO_CLUSTER_BUS** from the Bus name drop-down list, **TradeBrokerJSD** from the Queue name drop-down list, and **Nonpersistent** in the Delivery mode drop-down list, as shown in Figure A-28.

![Figure A-28](image)

*Figure A-28  A new queue is created for the default messaging*

25. Click **OK** to create the new queue.


   Select **Resources → JMS → JMS providers**.

27. From the Scope drop-down list, select **Cluster=PFCLUSTER**.
28. Click **Default messaging provider**.

29. Click **Topics**.

30. Click **New** to create a new topic.

31. In the Administration section, add **TradeStreamerTopic** to the Name and **jms/TradeStreameTopic** to the JNDI name, and in the Connection section add **TradeStreamerTopic** to the Topic name, select **PFITSO_CLUSTER_BUS** from the Bus name drop-down list, **Trade.Topic.Space** from the Topic space drop-down list, and **Nonpersistent** from the JMS delivery drop-down list, as shown in Figure A-29 on page 365.
32. Click **OK** to add the new topic.

33. Add an activation specification.

Select **Resources → JMS → JMS providers.**
34. From the Scope drop-down list, select **Cluster=PFCLUSTER** and click **Default messaging provider**.

35. Click **Activation specifications**.

36. Click **New** to create a new topic.

37. In the General properties section, add **TradeBrokerMDB** to the Name and **eis/TradeBrokerMDB** to the JNDI name, as shown in Figure A-30.

![Figure A-30](image-url)
38. In the Destination section, select **Queue** for the Destination Type, add **jms/TradeBrokerQueue** in the Destination JNDI name, and select **PFITSO_CLUSTER_BUS** from the Bus name drop-down list, as shown in Figure A-31.

![Figure A-31 Destination section for the Activation specification](image-url)
39. In the Additional section, select **TradeOSUserIDAuthData** from the Authentication alias drop-down list. In the Subscription Durability section, select **Nondurable** from the Subscription durability drop-down list. In the Advance section, select **In cluster** in the Shareable durable subscriptions, and **Default** for the Read ahead drop-down list, as shown in Figure A-32.

![Figure A-32](image)

Figure A-32  Additional properties, Subscription durability, and Advanced sections for the Activation specification

40. Click **OK** to add the new activation specification.

41. Click **New** to create another activation specification.
42. In the General properties section add `TradeStreamerMDB` to the Name and `eis/TradeStreamerMDB` to the JNDI name, as shown in Figure A-33.

![General Properties section for the TradeStreamerMDB activation specification](image_url)

*Figure A-33  General properties section for the TradeStreamerMDB activation specification*
43. In the Destination section, select Topic for the Destination Type, add jms/TradeStreamerTopic in the Destination JNDI name, select PFITSO_CLUSTER_BUS from the Bus name drop-down list, Auto-acknowledge from the Acknowledge mode, Bus member name from the Target type, and Preferred from the Target significance. In the Additional section, select TradeOSUserIDAuthData from the Authentication alias drop-down list, add 1 to the Maximum batch size, and add 10 to the Maximum concurrent endpoints, as shown Figure A-34.

Figure A-34  The destination and Additional properties sections for the TradeStreamerMDB activation specification
44. In the Subscription Durability section, select **Nondurable** from the Subscription durability drop-down list. In the Advance section, select **In cluster** in the Shareable durable subscriptions, and **Default** for the Read ahead drop-down list, as shown in Figure A-35.

![Figure A-35](image)

*Figure A-35  The Subscription durability and Advanced sections for the TradeStreamerMDB activation specification*

45. Click **OK** to add the new activation specification.
JDBC setup

Two JDBC connections need to be made, one for our SI Bus, which is running Derby, and one for our Trade V6.1 data, which is running on DB2.

1. Add a new JDBC provider for Derby.
   
   Select **Resources → JDBC → JDBC Providers**.

2. From the Scope drop-down list, select **Cluster=PFCLUSTER**, as shown in Figure A-36.

![Figure A-36: Selecting the scope for the JDBC providers](image)

3. Click **New** to create a new JDBC provider.
4. Select **Derby** from the Database type drop-down list, select **Derby JDBC Provider** from the Provider type drop-down list, select **Connection pool data source** from the Implementation type, and add **Derby JDBC Provider for Trade** to the Name field, as shown in Figure A-37.

![Figure A-37 Creating a Derby JDBC provider](image)

5. Click **Next**.
6. Enter the Derby library path in the Class path field. Derby should be located under the <WAS_HOME>/derby/lib directory, as shown in Figure A-38.

![Figure A-38 Adding the library path for the Derby JDBC provider](image)

7. Click **Next**.

8. Review the settings and click **Finish**, as shown in Figure A-39.

![Figure A-39 Confirmation window for the Derby JDBC provider](image)

1. Add a data source to the Derby provider.
   Select **Resources → JDBC → JDBC Providers**.
2. From the Scope drop-down list, select **Cluster=PFCLUSTER**.
3. Select the JDBC provider that was just added, **Derby JDBC Provider for Trade**.
4. Select **Data sources**.
5. Select **New**.

6. Add **PFITSO Cluster Bus DS** in the Name field and **jdbc/pfitso_cluster_bus_ds** in the JNDI name field, as shown in Figure A-40.

![Figure A-40   Adding a new data source for Derby](image)

7. Click **Next**.
8. Add the location and name of the Derby database in the Database name field. From the Derby configuration, we used <WAS_HOME>/derby/databases/PFClusterBusDB. Select the check box for **Use this data source in container managed persistence (CMP)**, as shown in Figure A-41.

![Figure A-41  Database specific properties for Derby](image1)

9. Click **Next**.

10. Click **Finish** on the confirmation window, as shown in Figure A-42.

![Figure A-42  Confirmation to add the new Derby data source](image2)
11. Check the connection to the Derby database by selecting the new data source and pressing the **Test connection** button, as shown in Figure A-43. A successful message should be returned if the data source was setup correctly.

![Figure A-43  Testing the new Derby data source connection](image)

12. Add a new JDBC provider for DB2.
   
   Select **Resources → JDBC → JDBC Providers**.

13. From the Scope drop-down list, select **Cluster=PFCLUSTER**.

14. Click **New**.
15. Select **DB2** from the Database type drop-down list, select **DB2 Universal JDBC Provider** from the Provider type drop-down list, select **XA data source** from the Implementation type, and add **DB2 JDBC Provider for Trade (XA)** for the Name field, as shown in Figure A-44.

![Figure A-44 Adding a new JDBC provider for DB2](image)

16. Click **Next**.
17. Enter the Class path information from the installation of DB2 and the Native library path, as shown in Figure A-45.

![Create a new JDBC Provider](image1)

**Figure A-45  Database path information for the DB2 JDBC provider**

18. Click **Next**.

19. On the Confirmation window, click **Finish** to create the DB2 JDBC provider, as shown in Figure A-46.

![Create a new JDBC Provider](image2)

**Figure A-46  DB2 JDBC provider confirmation window**
20. Add a data source to the DB2 JDBC provider.

   Select Resources → JDBC → JDBC Providers.

21. From the Scope drop-down list, select Cluster=PFCLUSTER.

22. Select the DB2 Provider created in the previous steps.

23. Click Data sources.

24. Click New.

25. Enter TradeDataSource for the Data source name, and jdbc/TradeDataSource for the JNDI name, as shown in Figure A-47.

Figure A-47   Adding a new datasource to the DB2 JDBC provider

26. Click Next.
27. We have used a type 4 driver for our testing. Enter the name of the DB2 location in the Database name field. The database location is case sensitive. Enter 4 in the Driver type field to indicate it is a type 4 driver. Enter the server name or IP address of the database in the Server name field, and listening port on the Port number field, as shown in Figure A-48.

![Figure A-48](image1.png)  
**Figure A-48**   The DB2 specific parameters for the JDBC provider

28. Click Next.

29. Click Finish on the confirmation window to create the new data source, as shown in Figure A-49.

![Figure A-49](image2.png)  
**Figure A-49**   DB2 data source confirmation window
Installation of the Trade V6.1 application

The installation of the Trade V6.1 application is similar to a normal application deployment, except that the bindings for database and JMS access must be updated to the bindings that have been defined previously in this chapter.

1. To install a new application, select Applications → Install New Application.

2. Select the trade.ear from either the local or remote systems, and select the Show me all installation options and parameters in the How do you want to install the application section, as shown in Figure A-50.

3. Click Next.

Note: The test connection button; however, this feature did not work with our testing using the WebSphere V6.1 Beta.
4. On the Bindings page, click **Next**, as shown in Figure A-51.

![Figure A-51 Bindings for the Trade application](image)

5. On the Security Warnings page, click **Next**, as shown in Figure A-52.

![Figure A-52 Security Warnings during the install of Trade is normal](image)

6. On the Select installation options window, select the **Distribute application** check box, select the **Deploy Enterprise Beans**, enter **Trade** in the
Application name field, select the **Create MBeans for resources** check box, and select **Deploy Web services** check box, as shown in Figure A-53.

![Install New Application](image)

**Figure A-53  Installation options for the Trade application**

7. Click **Next**.
8. On the Map modules to servers page, select the All modules and select the cluster and web server, if configured, in the Clusters and Servers box, and then click Apply, as shown in Figure A-54.

![Figure A-54  Mapping the modules to the servers for the Trade application](image)

9. On the Provide options for the EJB Deploy, select DB2UDBOS390_V8 for the Database type, as shown in Figure A-55.

![Figure A-55  EJB deployment options for the Trade application](image)

10. Click Next.
11. Accept the defaults and click **Next** for the JSP reloading options for the Web modules, as shown in Figure A-56.

![Figure A-56  JSP reloading options for the Trade application](image)

12. Accept the defaults and click **Next** for the Map shared libraries page, as shown in Figure A-57.

![Figure A-57  Shared libraries for the Trade application](image)
13. Accept the defaults and click **Next** for the Initialize parameters for servlets page, as shown in Figure A-58.

![Figure A-58 Initialization parameters for the servlets in the Trade application](image)

14. Accept the defaults and click **Next** on the Bind listeners for message-driven beans page, as shown in Figure A-59.

![Figure A-59 Bindings for the MDB listeners for the Trade application](image)
15. Accept the defaults and click **Next** for the Provide JNDI names for beans page, as shown in Figure A-60.

![Figure A-60](image1)

**Figure A-60**  JNDI names for the EJBs for the Trade application

16. Select the defaults and click **Next** for the Map EJB references to beans page, as shown in Figure A-61.

![Figure A-61](image2)

**Figure A-61**  Mapping the EJB references to beans for the Trade application
17. Accept the defaults and click **Next** for the Bind message destination references to administered objects page, as shown in Figure A-62.

![Figure A-62 Binding the message destinations to administered objects for the Trade application](image)

**Figure A-62**  Binding the message destinations to administered objects for the Trade application
18. Select the **TradeDataSourceAuthData** in the Use default method (many-to-one mapping) and select the section below with the EJB modules. Afterwards, click **Apply**, as shown in Figure A-63.

![Diagram of TradeDataSourceAuthData configuration]

**Figure A-63**  Mapping authentication data entry to the CMPs in the Trade application

19. Click **Next**.
20. In the javax.sql.DataSource section, select the TradeDataSourceAuthData from the Use default method (many-to-one mapping) drop-down list and select the modules and press the Apply button, as shown in Figure A-64.

![Figure A-64 Setting the authorization for the JDBC data sources for the Trade application](image)

**Figure A-64** Setting the authorization for the JDBC data sources for the Trade application

21. Click Next.

22. Leave the defaults on the Map virtual hosts for the Web modules page and click Next, as shown in Figure A-65.

![Figure A-65 Mapping the virtual hosts to the web modules for the Trade application](image)

**Figure A-65** Mapping the virtual hosts to the web modules for the Trade application
23. Leave the defaults for the context root and click **Next**, as shown in Figure A-66.

![Figure A-66  The context root for the Trade application](image)

24. Click **Next** on the WebSphere options page.

25. Click **Finish** to start the deploy of the application, as shown in Figure A-67.

![Figure A-67  Deployment summary for the Trade application](image)

26. After the successful deployment, save the configuration.
27. Now the application can be started. Select **Applications → Enterprise Applications**. Select the **Trade** application check box and click the **Start** button, as seen in Figure A-68.

![Figure A-68 Starting the Trade application](image1)

The Trade application is accessible from the Web browser using http://<host>:<port>/trade.

![Figure A-69 The Trade application viewed from a Web browser](image2)

**Note:** Check for error messages in the Servants on z/OS to determine if any errors have occurred in startup or configuration of the Trade application.
Additional material

This IBM Redbooks publication refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this book is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:


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 book form number, SG247269.
Using the Web material

The additional Web material that accompanies this book includes the following files:

<table>
<thead>
<tr>
<th>File name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMeter.zip</td>
<td>Zipped file with sample JMeter scripts</td>
</tr>
<tr>
<td>gc analyzer.zip</td>
<td>Zipped file with our GCAnalyzer tool</td>
</tr>
</tbody>
</table>
# Abbreviations and acronyms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>BIRT</td>
<td>Business Intelligence and Reporting Tools</td>
</tr>
<tr>
<td>CCI</td>
<td>Common Client Interface</td>
</tr>
<tr>
<td>CF</td>
<td>Connection Factory</td>
</tr>
<tr>
<td>CICS</td>
<td>Customer Information Control System</td>
</tr>
<tr>
<td>CICS TG</td>
<td>CICS Transaction Gateway</td>
</tr>
<tr>
<td>CMP</td>
<td>Container Managed Persistence</td>
</tr>
<tr>
<td>COMMAREA</td>
<td>Communication area</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture</td>
</tr>
<tr>
<td>EAR</td>
<td>Enterprise Application Archive</td>
</tr>
<tr>
<td>EIS</td>
<td>Enterprise Information System</td>
</tr>
<tr>
<td>EJB</td>
<td>Enterprise Java Bean</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>IBM</td>
<td>International Business Machines Corporation</td>
</tr>
<tr>
<td>IC</td>
<td>Initial Context</td>
</tr>
<tr>
<td>IIOP</td>
<td>Internet Inter-ORB Protocol</td>
</tr>
<tr>
<td>IMS</td>
<td>Information Management System</td>
</tr>
<tr>
<td>ITSO</td>
<td>International Technical Support Organization</td>
</tr>
<tr>
<td>J2EE</td>
<td>Java 2 Enterprise Edition</td>
</tr>
<tr>
<td>JACC</td>
<td>Java Authorization Contract for Containers</td>
</tr>
<tr>
<td>JAF</td>
<td>Java Beans Activation Framework</td>
</tr>
<tr>
<td>JAR</td>
<td>Java Archive</td>
</tr>
<tr>
<td>Java EE</td>
<td>Java Enterprise Edition</td>
</tr>
<tr>
<td>JAXR</td>
<td>Java API for XML Registries</td>
</tr>
<tr>
<td>JAX-RPC</td>
<td>Java API for XML-Based Remote Procedure Calls</td>
</tr>
<tr>
<td>JAX-WS</td>
<td>Java API for XML Web Services</td>
</tr>
<tr>
<td>JCA</td>
<td>J2EE Connector Architecture</td>
</tr>
<tr>
<td>JDBC</td>
<td>Java Database Connectivity</td>
</tr>
<tr>
<td>JDK</td>
<td>Java Development Kit</td>
</tr>
<tr>
<td>JMS</td>
<td>Java Message Service</td>
</tr>
<tr>
<td>JNDI</td>
<td>Java Naming and Directory Interface</td>
</tr>
<tr>
<td>JRE</td>
<td>Java Runtime Environment</td>
</tr>
<tr>
<td>JSF</td>
<td>Java Server Faces</td>
</tr>
<tr>
<td>JSP</td>
<td>JavaServer Pages</td>
</tr>
<tr>
<td>JSTL</td>
<td>JSP Standard Tag Library</td>
</tr>
<tr>
<td>JTA</td>
<td>Java Transaction API</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>MDB</td>
<td>Message Driven Bean</td>
</tr>
<tr>
<td>MQ</td>
<td>Message Queuing</td>
</tr>
<tr>
<td>ODBC</td>
<td>Open Database Connectivity</td>
</tr>
<tr>
<td>ORB</td>
<td>Object Request Broker</td>
</tr>
<tr>
<td>PS</td>
<td>Publish-and-Subscribe</td>
</tr>
<tr>
<td>PTP</td>
<td>Point-To-Point</td>
</tr>
<tr>
<td>RAD</td>
<td>Rational Application Developer</td>
</tr>
<tr>
<td>RMI</td>
<td>Remote Method Invocation</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Call</td>
</tr>
<tr>
<td>RRS</td>
<td>Resource Recovery Services</td>
</tr>
<tr>
<td>SAAJ</td>
<td>SOAP with Attachments API for Java</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>Abbr</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>SQL</td>
<td>Standard Query Language</td>
</tr>
<tr>
<td>SQLJ</td>
<td>SQL in Java</td>
</tr>
<tr>
<td>StAX</td>
<td>Streaming APIs for XML Parsers</td>
</tr>
<tr>
<td>TPS</td>
<td>Transactions Per Second</td>
</tr>
<tr>
<td>TPTP</td>
<td>Test and Performance Tools Platform</td>
</tr>
<tr>
<td>WAR</td>
<td>Web Application Archive</td>
</tr>
<tr>
<td>WAS</td>
<td>WebSphere Application Server</td>
</tr>
<tr>
<td>WD4z</td>
<td>WebSphere Developer for IBM System z</td>
</tr>
<tr>
<td>WS</td>
<td>WebService</td>
</tr>
<tr>
<td>WSAD IE</td>
<td>WebSphere Application Developer Integration Edition</td>
</tr>
</tbody>
</table>
Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this IBM Redbooks publication.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 400. Note that some of the documents referenced here may be available in softcopy only.

- *CICS Transaction Gateway for z/OS Version 6.1*, SG24-7161
- *DB2 for z/OS and WebSphere: The Perfect Couple*, SG24-6319
- *IMS Connectivity in an On Demand Environment: A Practical Guide to IMS Connectivity*, SG24-6794
- *Java Message Service (JMS) Security on z/OS*, REDP-4203
- *Problem Determination for WebSphere for z/OS*, SG24-6880
- *System Programmer’s Guide to: Workload Manager*, SG24-6472
- *WebSphere for z/OS to CICS and IMS Connectivity Performance*, REDP-3959
- *WebSphere for z/OS Connectivity Architectural Choices*, SG24-6365
- *WebSphere for z/OS V6 Connectivity Handbook*, SG24-7064
- *z/OS Distributed File Service zSeries File System Implementation z/OS V1R7*, SG24-6580

Other publications

These publications are also relevant as further information sources:

- *Application Programming and SQL Guide*, SC26-9933
- *HiperSockets: A Dramatic Increase in Networking Bandwidth* at the following site:
  
▶ JMeter User Guide:
   http://jakarta.apache.org/jmeter/usermanual/remote-test.html
▶ UNIX System Services Command Reference, SA22-7802
▶ WebSphere for z/OS Version 6 -- Understanding the HTTP Web Server Plugin, PRS1467
▶ z/OS HTTP Server Planning, Installing and Using, SC34-4826
▶ z/OS MVS System Management Facilities (SMF), SA22-7630
▶ z/OS MVS Planning: Workload Management, SA22-7602
▶ z/OS Resource Measurement Facility Report Analysis, SC33-7991
▶ HiperSockets Implementation Guide, SG24-6816

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
64-bit mode, storage 236

A
abend code
EC3 57
abend EC3/0413000x 173
Accessing EIS
DB2 302
AutoCommit 307
connection context object 317
collection pooling 309
data type mapping 303
datasource, defining 310
DATESTAMP 311
DB2 built-in functions, using 308
dynamic statement caching 305
EXPLAIN PLAN 319
getString() method 312
getxxx() method 311
KEEP_CONNECTION 318
numbers, storing 308
PLAN, rebinding 320
pooled connection, using 310
resources, releasing 308
row prefetching 314
SELECT and UPDATE 307
serialized profile 317
SQLJ 304
statement caching 313
TIMESTAMP 311
transactions 312
update batching 313
JCA
connection factor, caching 320
collection pooling 325
connection usage, azy association 326
connection usage, in JCA 1.0 325
connection usage, in JCA 1.5 325
CTG
data conversion 329
Java Message Service (JMS) 330
AUTO_ACKNOWLEDGE 332
CLIENT_ACKNOWLEDGE 332
collection best practices 330
ConnectionConsumer 331
delivery mode value 334
destination 333
DUPS_OK_ACKNOWLEDGE 332
Message object 336
MessageConsumer object 336
MessageListener interface 336
onMessage method 336
processing messages concurrently 331
QueueSession object 332
session 332
session, acknowledgement mode 332
TopicSession object 332
transactional messages 332
type of message 336
managed environment 323
non-managed environment 323
transactions 323
transactions, lazy enlistment 327
JCA, IMS Connect 329
affinity, requests with 227
Agent Controller 13
Agent Controller, difference between RPT and TPTP 74
Apache JMeter (JMeter) 46
Comma Separate Values (CSV) file 52
configuration elements 48
custom Java classes 51
distributed architecture 47
Pre and Post Processors 50
protocols supported 47
Remote Method Invocation (RMI) protocol 47
test script format 47
Thread Groups 50
Timers 50
User Parameters 49
Application profiling
fractional analysis 72
areas 72
statistical analysis 72
Application Server Controller 222
C

cachespec.xml 177
CB Service Class 16
CICS Transaction Gateway for z/OS Version 6.1 257
CICS Transaction Gateway on z/OS
CTG_PIPE_REUSE environment variable, in WAS 259
EXCI pipes 259
local mode
  RECEIVECOUNT in CICS region 259
  threads available 258
  threads available, in Admin Console 259
  Workload profile, in WAS 259
modes of operation 258
remote mode
  Gateway daemon 260
  Gateway daemon, Connection Manager threads 260
  Gateway daemon, pools of threads 260
  Gateway daemon, thread parameters 261
  Gateway daemon, Worker threads 260
  RECEIVECOUNT in CICS region 260
Client container 274
com.ibm.CORBA.FragmentSize 182
com.ibm.CORBA.iiop.noLocalCopies 182
Comma Separated Value (CSV) file
  Comma Separated Value (CSV) file 41
Common Client Interface (CCI) 302
Compensation Service 234
configuration HFS, mounting R/O 218
ConnectionFactory 250
container 274
Controllers 222
Coupling Facility (CF) 233

D

Database Request Module (DBRM) 317
Deployment Manager Controller 222
Derby 12
destroy() method 290
dynamic MVS console commands 139

capabilities (BIRT) 73
Eclipse Infocenter 76
Eclipse Test & Performance Tools Platform (TPTP) 71
  Agent Controller 74
  standard ports, changing 80
  Agent Controller, permissions 79
  Agent Controller, starting 81
  architecture 75
  BIRT, full installation package 77
  components 75
  configuration, used in this Redbook 76
  filters 75
  installation, Agent Controller 79
  installation, plug-in 77
  overhead 73
  overview 72
  ports 74
  prerequisites 76
  prerequisites, host 77
  prerequisites, workstation 76
  profiling approaches supported 72
  screen components 83
  WAS servant, JVM property 81
Edge side include caching
  edge side include caching 178
EJB container 274
enclave 222
Enterprise Java Beans 275
Enterprise JavaBeans 274
Entity Bean 276

G
gencon 195
Global Resource Serialization (GRS) 235
Global Resource Serialization (GRS), options 235
Global Security 11

H
heap analysis 129
heap dump 150
high CPU utilization 57
high response time 56
HiperSockets 221
httpd.conf 162

Eclipse Business Intelligence and Reporting Tools
<table>
<thead>
<tr>
<th>IBM HTTP Server</th>
<th>IBM Rational Agent Controller (RAC), level required for RPT 38</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNS lookup 161</td>
<td>IBM Rational Software Development Platform (Rational SDP) 38</td>
</tr>
<tr>
<td>httpd.conf 162</td>
<td>IBM Tivoli Composite Application Manager for WebSphere (ITCAM for WebSphere) 109</td>
</tr>
<tr>
<td>parameters</td>
<td>IBM Tivoli Composite Application Monitor for WebSphere (ITCAM for WebSphere) administration 116</td>
</tr>
<tr>
<td>MaxActiveThreads 161</td>
<td>Configuration Library 119</td>
</tr>
<tr>
<td>plug-in, for WebSphere Application Server 162</td>
<td>Data Collector, configuring 119</td>
</tr>
<tr>
<td>plugin-cfg.xml 162</td>
<td>Enterprise Overview page 117</td>
</tr>
<tr>
<td>threads, usage 161</td>
<td>L1 (Production Mode) level 119</td>
</tr>
<tr>
<td>WAS plug-in settings 162</td>
<td>L2 (Problem Determination Mode) level 120</td>
</tr>
<tr>
<td>blocking connect 166</td>
<td>monitoring level, changing 120</td>
</tr>
<tr>
<td>ConnectTimeout 166</td>
<td>new user, adding 116</td>
</tr>
<tr>
<td>ConnectTimeout, no value 166</td>
<td>role, adding 117</td>
</tr>
<tr>
<td>ConnectTimeout, recommended value 167</td>
<td>schedule 121</td>
</tr>
<tr>
<td>ConnectTimeout, value greater than 0 167</td>
<td>server group, adding 117</td>
</tr>
<tr>
<td>ConnectTimeout, value of 0 166</td>
<td>user access 116</td>
</tr>
<tr>
<td>ConnectTimeout, viewing in admin console 167</td>
<td>Data Collectors 110, 113</td>
</tr>
<tr>
<td>LoadBalance 165</td>
<td>architecture 115</td>
</tr>
<tr>
<td>LoadBalance, default value 165</td>
<td>Command Agent 113</td>
</tr>
<tr>
<td>LoadBalance, Round Robin option 165</td>
<td>components 113</td>
</tr>
<tr>
<td>LoadBalance, recommended value 165</td>
<td>Event Agent 113</td>
</tr>
<tr>
<td>LoadBalance, Round Robin option 165</td>
<td>platforms 113</td>
</tr>
<tr>
<td>LoadBalance, viewing in admin console 166</td>
<td>probes 114</td>
</tr>
<tr>
<td>LoadBalanceWeight 168</td>
<td>resources to gather data 114</td>
</tr>
<tr>
<td>LoadBalanceWeight, viewing in admin console 169</td>
<td>Windows prerequisites 113</td>
</tr>
<tr>
<td>LogLevel 163</td>
<td>z/OS 113</td>
</tr>
<tr>
<td>LogLevel, default value 163</td>
<td>IBM zSeries Application Assist Processor (zAAP) support 115</td>
</tr>
<tr>
<td>LogLevel, recommended value 163</td>
<td>information 110</td>
</tr>
<tr>
<td>LogLevel, values 163</td>
<td>Managing Server 110</td>
</tr>
<tr>
<td>MaxConnections 167</td>
<td>Archive Agents 111</td>
</tr>
<tr>
<td>MaxConnections, default value 167</td>
<td>components 111</td>
</tr>
<tr>
<td>MaxConnections, recommended value 167</td>
<td>databases supported 111</td>
</tr>
<tr>
<td>MaxConnections, viewing in admin console 168</td>
<td>Global Publishing Servers 111</td>
</tr>
<tr>
<td>RefreshInterval 162</td>
<td>Kernel 111</td>
</tr>
<tr>
<td>RefreshInterval, default value 162</td>
<td>Message Dispatcher 111</td>
</tr>
<tr>
<td>RefreshInterval, recommended value 162</td>
<td>our setup 112</td>
</tr>
<tr>
<td>RetryInterval 164</td>
<td>platforms 110</td>
</tr>
<tr>
<td>RetryInterval, default value 165</td>
<td>Polling Agents 111</td>
</tr>
<tr>
<td>RetryInterval, lowering 165</td>
<td>Publishing Servers 111</td>
</tr>
<tr>
<td>RetryInterval, recommended value 165</td>
<td>sizing 112</td>
</tr>
<tr>
<td>RetryInterval, viewing in admin console 165</td>
<td>Visualization Engine 111</td>
</tr>
<tr>
<td>WAS plug-in settings, setting in Admin Console 163</td>
<td>Windows prerequisites 111</td>
</tr>
<tr>
<td>IBM HTTP Server, parameters</td>
<td>monitoring</td>
</tr>
<tr>
<td>MaxActiveThreads, recommended value 162</td>
<td>Enterprise Overview 124</td>
</tr>
<tr>
<td>IBM Rational Agent Controller (RAC) 38</td>
<td></td>
</tr>
</tbody>
</table>
Performance Monitoring and Best Practices for WebSphere on z/OS

Group Overview 124
L3 (Tracing Mode) level 120
problems with memory, examining 129
tracking actions 122
Server Detail View 127
Server Group 124
Server Overview 126
Server Overview 126
SNMP message 122
System Activity Display 127
System Resource Overview, components examined 128
System Statistics Overview 126
trap 122
trap, example 123
monitoring levels 119
performance analysis 133
Application Reports 133
Server Reports 136
Top Reports 134
sources of data 110
thresholds 110
IBM Trade V6.1 21
application architecture 22
main services 23
IMS Connect
modes of operation 262
socket parameters 263
sockets, maximum number 263
IMS Connect Version 2.2 261
IMS Connector for Java 261
ITSO Trader V6.1 23
application architecture 25
ITSO Trader V6.1 service interfaces 26

J
Java coding best practices
aggregating business objects 282
algorithms 278
ArrayList 278
asynchronous processing 283
collection type 278
data Caching 282
data structures 282
EJB 291
two CMP entity Beans 296
access intent policies 298
access intent service 298
caching 298
entity Bean 293
Fast Lane Reader design pattern 295
isolation level 298
local interfaces 291
stateful session beans, removing 299
transaction settings 296
exceptions 280
good candidates for caching 283
Java Naming and Directory Interface (JNDI) 283
LinkedList 278
logging 279
printStackTrace() 280
String 279
StringBuffer 279
System.err.println 280
System.out.println 280
toString() 279
Web
custom tags, in JSP 289
includes, in JSPs 288
JSP auto reloading 290
scope, of bound objects in JSPs 287
session caching 290
session size 284
session.invalidate() method 285
wrapping and accessing EJB entity beans in
EJB session beans 296
XML parsing 283
Document Object Model (DOM) 283
object serialization 284
Simple API for XML (SAX) 283
Java Development Kit (JDK), in WebSphere 11
Java Message Service (JMS) 250
BOQNAME 253
BOTHRESH 253
connection factory settings 254
collection pool 254
maximum connections 254, 256
maximum connections, for BINDINGS mode
connections 255
maximum connections, for CLIENT mode
connections 255
session pool 255
Listener Port 251–252
maximum messages parameter 252
maximum retries 253
listener port 250
Message Listener Service 251
queue connection 250
queue session 250
session pool 250
WebSphere MQ settings
   DEFSOPT 257
   INDXTYPE 256
   SHARE 257
Java Virtual Machine
   heap usage 57
Java Virtual Machine (JVM)
   Garbage Collection (GC)
      Compact time 206
      interval 200
      Mark time 205
      mark time 205
      phases 196
      policies 195
      Sweep time 206
      sweep time 206
   heap 193
      Large Object Area (LOA) 194
      Large Object Area (LOA), adjusting 194
      Large Object Area (LOA), disable 195
      Small Object Area (SOA) 194
   heap parameters, changing in Admin Console 194
   heap size, default maximum 193
   heap size, initial 193
      changing 193
   heap size, maximum
      changing 193
   heap, expansion 193
   heap, shrinkage 194
   Just In Time (JIT) 215
   VerboseGC log 197
   VerboseGC, switching on 197
   Xloainitial 197
   Xloamaximum 194
Java Virtual Machine Tool Interface (JVMTI) 95
JavaDatabase Connectivity (JDBC) 239
JavaServer Pages 274
JavaServer Pages (JSP) 275
jsptCompileClasspath 187

L
Language Environment (LE) 234
   HEAPCHK(ON) 235

P
Performance Monitoring Infrastructure (PMI) 92
   counters for J2C connectors 264
   counters for JDBC 265
   Custom option 93
   data used 92
   enablement 93
   enablement, in admin console 94
   JVMTI, using 95
   settings, custom 95
   settings, default 94
   statistic sets 93
Performance Monitoring Infrastructure (PMI), with J2C and JDBC 263
plugin-cfg.xml 162
problems with memory, examining 129
protocol_http_backlog 221
protocol_https_backlog 221
protocol_iiop_backlog 221
protocol_ssl_backlog 221

R
Rational Performance Tester (RPT) 37
architecture 38
creating reports from WSAM/ITCAM 42
custom Java classes 44
Data Pool 40
detailed architecture 39
IBM Rational ClearCase LT 44
protocols supported 38
RPT Test Scheduler, user groups 43
Test Scheduler 41

reason code
0413000x 57
Redbooks Web site 400
  Contact us xv
REGION parameter, in JCL 236
Report Class report 16
Report Classes 15
Resource Analyzer 96
Resource Measurement Facility (RMF) 58
  Workload Activity Report 230
Resource Recovery Services (RRS) 233
  logstream 233
  logstream, size 234
response time 270
RMF Monitor III 60
RMF Performance Monitoring Java Technology Edition (RMF PM) 58–59
  DataView, creating 68
  DataView 60
  Distributed Data Server (DDS) 60
  Distributed Data Server, installation 62
    access to BPX.DAEMON facility 62
    definitions for RACF 62
    GPMSRV00 parmlib member 63
    MINTIME and SYNC options 63
    ports 63
    PROGRAM CONTROL 62
  Distributed Data Server, starting 64
  Distributed Data Server, trace 64
exporting graph data 69
installing, on Windows 65
monitoring, starting 68
PerfDesk, creating 67
PerfDesks 60
prerequisites, on host 61
spreadsheet 60
sysplex 60
RRS Transactional property 323
RUNSTATS 320

S
sample workload
  business transaction scenarios 26
  user profiles 26
scalability 271
serialized profile 317
servants 223
Service Class definition 16
Service Class report 16
Service Locator 283
servlet 274–275
servlet init() method 290
Session Bean 276
SMF Browser 58
SMF Dump program (IFASMFDP) 71
SMP/E install HFS 218
SMP/E install HFS, mounting R/O 218
SQLJ profile 317
stateful session Bean 276
stateless session Bean 276
storage 236
storage constraints, checking 236
stored procedure 245
subpool 195
SVC dump 150
SVCDUMP 57
System Management Facilities (SMF) 70
  70-79 records 231
  92 record 218
  Activity records 70
  Interval records 70
  record type 120 70
  record type 120, enable and collect 70
  record type 120, subtypes 70
System Management Facility (SMF) 58

T
TCP/IP
  DNS configuration 221
  DNS configuration, TTL (Time To Live) value
221
finwait2 status 220
listen backlog 221
listen backlog variables, in Admin Console 221
TCPIP.DATA
   RESOLVERTIMEOUT 221
TCPIP.PROFILE 220
   FINWAIT2TIME 220
   NODELAYACKS 220
   SOMAXCONN 221
   TCPRCVBUFSIZE 220
   TCPSENDBUFSIZE 220
TCP/IP sockets 220
temporal affinity 227
The Java 2 Platform, Enterprise Edition (J2EE) 274
thread dump 132, 150
throughput 270
timeout 57
Tivoli Performance Viewer (TPV) 96
   Advisor 100
   log settings 99
   monitoring 105
   Network Deployment setup 97
   performance advisors 101
      resources 102
   performance modules 103
   performance modules, viewing data 103
   Performance Monitoring Infrastructure (PMI) data, using 97
   PMI data, scaling 104
   settings, configuring 97
   single server setup 97
   summary reports 102
   summary reports, viewing 103
   topologies 97
   user settings 98
      buffer size 98
      refresh rate 98
Trade application 342
   Cloudscape 342
   DB2, setup 344
   deployment of the application 382
   Derby, creating 344
   Derby, Data Definition Language (DDL) for SI-Bus 343
   Derby, setup 343
   J2C authentication 345
   Java Message Service (JMS) setup 355
   JDBC setup 372
   SI Bus, creating 347
   WebSphere clusters 344
U
Uniform Resource Identifier (URI) 276
UNIX System Services (USS) 218
   adding a file to the cache 218
   file descriptors in use, monitoring 219
   filecache command 218
   MAXFILEPROC 218–219
   MAXSOCKETS 218–219
   socket usage, monitoring 219
   SYS1.PARMLIB(BPXPRMxx) member 218
V
verbose Garbage Collection (GC) 57
   overhead 58
   virtual storage 236
Virtual Users (VU) 10
W
WAS connectivity
   Aged Timeout 241–242
   connection factory 239
   Connection Manager 239
   connection pool, properties 240
   connection pooling 239
   Connection Timeout 240
   ConnectionWaitTimeout exception 241
   J2C connectors 239
   Java DataBase Connectivity
      callable statement 245
      connection pool properties, setting in Admin Console 248
      data source tuning 244
      DB2 system parameters, for DB2 dynamic statement caching 246
      DB2 threads 249
      dynamic statement cache, in WAS 246
      dynamic statement caching, in DB2 246
      JDBC Type 249
      keepDynamic custom property 247
      KEEPDYNAMIC=YES BIND option 248
      prepared statement 245
      specific properties 243
      statement cache size 244
      statement cache, largest value 245
Java Message Service (JMS) 239
Java Database Connectivity (JDBC) 239
Maximum Connections 240–241
maximum connections 240
Minimum Connections 241
Purge Policy 243
Purge Policy, EntirePool 243
Purge Policy, FailingConnectionOnly 243
Reap Time 242
Unused Timeout 242
Unused Timeout setting 241
Web container 274
Web service 276
WebSphere Application Server
  versions 9
WebSphere Application Server (WAS)
  abend EC3/0413000x 173
  Automatic Restart setting 180
  bouncing server 174
  cache policy file 177
  cachespec.xml 177
  caching, types 178
  cipher suites 177
  command caching 178
  control_region_dreg_on_no_srs variable 175
  control_region_timeout_delay variable 175
  Dynamic Application Reloading 184
  Dynamic Application Reloading, disabling 184
  Dynamic Cache Service 177
  dynamic cache service 183
  dynamic caching, enabling 178
  EJB container
    cleanup interval 189
  EJB container cache size 189
  EJBROLES 176
  HA Manager 183
    workload management routing 183
  HA Manager, disabling 183
    sample script for wsadmin 183
  Hot Deployment 184
  Java 2 Security 176
  JavaServer Pages (JSPs), pre-compiling 186
  JRAS tracing 170
  JSP batch compiler tool 187
  jspCompileClasspath 187
  LogStream variable 179
  LogStreams, compressing 179
  Maximum Number of Instances 173
  memory-to-memory replication 183
  Minimum Number of Instances 172
  Monitoring Policy Ping interval 179
  Monitoring Policy Ping interval, default 179
  number of servants 172
  Object Request Broker (ORB) 182
    passing parameters 182
  portlet fragment caching 178
  protocol_accept_http_work_after_min_srs variable 174
  protocol_http_timeout_output_recovery variable 173
  protocol_http_timeout_output_recovery variable, setting in Admin Console 174
  reloadding, EJB and WAR modules 184
  reloadding, JSPs 185
  Secure Sockets Layer (SSL) 177
  Security Attribute Propagation (SAP) 177
  servlet caching 178
  singleton failover 183
  smpe.install.root variable 218
  synchronization interval 180
  timeout abend, avoiding 173
  timeout grace period 175
  tracing 170
    ras_trace_basic 170
    ras_trace_BufferCount 170
    ras_trace_BufferSize 170
    ras_trace_defaultTracingLevel 170
    ras_trace_detail 170
    ras_trace_outputLocation 170
    timeout 173
  URL invocation cache 187
  URL invocation cache, JVM heap size 187
  was.install.root variable 218
  Web services client cache 178
  WLMStatefulSession variable 172
  Workload Profile 171
  Workload Profile, options 171
  WebSphere settings, general 17
  WebSphere Studio Workload Simulator (WSWS)
    architecture 34
    capture function 35
    Controller 34
    Engine 34
    graphing function 36
    GUI 35
    overview 34
    playback function 35
    protocols supported 37
WLM settings 14
wlm_maximumSRCount 173
wlm_minimumSRCount 172
worker (application) threads 171
workload 30
Workload Manager (WLM) 222
  classification 222
  classification, of controllers and daemons 222
  classification, of servants 223
  classification, of work running under an enclave 223
  classification.xml 224
  Collection Name (CN) 223
  Maximum Number of Instances property 224
  Minimum Number of Instances property 224
  Performance Index (PI) 58
  request classification, monitoring 225
  response goal percentile 226
  Server Instance Name (SI) 223
  Service Class 58
  Service Class, example 226
  SRCount variables 225
  temporal affinity 227
  Transaction Class (TC) 223
  User ID (UI) 223
  velocity goal 226
  wlm_classification_file variable 224
  wlm_maximumSRCount variable 224
  WLMStatefulSession variable 228–229
  WLMStatefulSession variable, equivalent in WAS Version 5.1 229
  WLMStatefulSession variable, setting in Admin Console 229
  WSOOWNER 347

X
  XA Partner Log 234
  XML C++ Parser 13

Z
  zFS 218
Implementing a WebSphere Application Server environment on z/OS and deploying applications to it in a manner that delivers the best performance is not a trivial task. There are many parameters that can be influenced and applications can be written in many different ways.

This IBM Redbooks publication will give you a structure you can use to set up an environment that is tuned to meet best performance and at the same can be monitored to catch eventual performance bottlenecks. We also pay attention to workload testing using a variety of tools and best practices in writing and deploying applications.