Monitoring WebSphere V5.1 Application Performance on z/OS

WebSphere V5.1 on z/OS performance methodology

Tivoli, OMEGAMON, WSAM

Monitor and troubleshoot production performance of WebSphere V5.1 on z/OS

Patrick C Ryan
G Michael Connolly
Phil Anselm
Charles Neidigh
Raymond Sun
Theresa Tai
Michael Weintraub

ibm.com/redbooks
Monitoring WebSphere Application Performance on z/OS

April 2006
Note: Before using this information and the product it supports, read the information in “Notices” on page xi.


This edition applies to WebSphere Application Server for z/OS V5.1

This document created or updated on June 20, 2006.

© Copyright International Business Machines Corporation 2006. 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 ........................................................................................................... xi
Trademarks .................................................................................................... xii

Preface .......................................................................................................... xiii
The team that wrote this redbook. ................................................................. xiii
Become a published author ...................................................................... xiv
Comments welcome ................................................................................. xv

Part 1. Theory and methodology ................................................................. 1

Chapter 1. Theory of performance monitoring ........................................... 3
1.1 Introduction ............................................................................................... 4
1.2 Overview ................................................................................................. 4
1.3 Performance metrics ............................................................................ 5
1.4 Components impacting performance .................................................... 11
  1.4.1 Application performance factors .................................................... 11
  1.4.2 WebSphere Application Server for z/OS ....................................... 13
  1.4.3 z/OS environment ........................................................................ 15
1.5 Setting expectations .............................................................................. 16

Chapter 2. Methodology of performance monitoring ................................. 19
2.1 Overview ................................................................................................. 20
2.2 Performance monitoring cycle ............................................................... 21
  2.2.1 Initial installation ......................................................................... 21
  2.2.2 Setting a baseline ........................................................................ 21
  2.2.3 Monitoring .................................................................................. 23
  2.2.4 Evaluating .................................................................................. 25
  2.2.5 Modifying .................................................................................. 25
  2.2.6 Summary .................................................................................. 26

Part 2. Monitoring tools ............................................................................. 27

Chapter 3. Basic tools ................................................................................ 29
3.1 System Management Facilities records ............................................... 30
  3.1.1 The z/OS SMF Summary Viewer .................................................. 30
3.2 Resource Measurement Facility ........................................................... 31
  3.2.1 RMF Monitor III displays .............................................................. 31
  3.2.2 RMF PM .................................................................................. 32
  3.2.3 RMF Monitor I reports ................................................................. 33
### Contents

7.2.1 Sysplex summary ................................................. 77
7.2.2 Response time distribution ..................................... 79
7.2.3 Work Manager delays .............................................. 80
7.2.4 System information ................................................ 81
7.3 RMF reports for baseline scenario ................................. 82
  7.3.1 Workload activity report ....................................... 83
  7.3.2 Workload activity transactions ................................. 84
  7.3.3 Workload activity WASPC ....................................... 86
  7.3.4 System summary .................................................. 87
7.4 Tivoli Performance Viewer for baseline scenario ................. 88
  7.4.1 Servlets .......................................................... 88
  7.4.2 JVM runtime ...................................................... 89
  7.4.3 Object Request Broker pool summary ......................... 90
7.5 OMEGAMON for baseline scenario .................................. 91
  7.5.1 WLM service class resources ................................. 92
  7.5.2 Servant regions .................................................. 93
  7.5.3 Garbage collection activity .................................... 94
  7.5.4 Garbage collection history ..................................... 95
  7.5.5 Web applications ................................................ 96
  7.5.6 Web applications: Ten worst average response times ........ 97
  7.5.7 EJB containers ................................................... 98
  7.5.8 EJB method invocations ....................................... 99
7.6 WSAM for baseline scenario ........................................ 100
  7.6.1 Enterprise overview ............................................ 101
  7.6.2 Server overview .................................................. 102
  7.6.3 Servant regions overview ...................................... 102
  7.6.4 Server statistics overview ..................................... 103
  7.6.5 WLM service class period detail ............................... 104
  7.6.6 Number of requests versus average response time .......... 105
  7.6.7 Number of requests versus total garbage collection time .. 106

### Chapter 8. Scenario 1: Write to hierarchical file system file

8.1 Overview .......................................................... 107
8.2 RMF displays for scenario 1 ....................................... 108
  8.2.1 Sysplex summary ............................................... 108
  8.2.2 Response time distribution ..................................... 110
  8.2.3 Work Manager delays .......................................... 111
  8.2.4 System information ............................................. 112
8.3 RMF reports for scenario 1 ....................................... 113
  8.3.1 Workload activity report ....................................... 114
  8.3.2 Workload activity transactions ............................... 115
  8.3.3 System summary report ........................................ 117
8.4 TPV for scenario 1 ................................................... 117
Chapter 9. Scenario 2: Excessive objects .......................... 127
9.1 Overview .................................................... 128
9.2 RMF displays for scenario 2 .................................. 128
  9.2.1 Sysplex summary ........................................ 128
  9.2.2 Response time distribution ............................... 130
  9.2.3 Work Manager delays ................................... 131
  9.2.4 System information ....................................... 132
9.3 RMF reports for scenario 2 .................................... 134
  9.3.1 Workload activity report ................................. 134
  9.3.2 Workload activity transactions ......................... 135
  9.3.3 System summary report ................................ 136
9.4 TPV for scenario 2 ........................................... 137
  9.4.1 Java virtual machine runtime ........................... 137
9.5 OMEGAMON for scenario 2 ................................... 138
  9.5.1 Workload Manager (WLM) service class resources .... 138
  9.5.2 Address space bottlenecks ............................. 139
  9.5.3 Servant regions .......................................... 140
  9.5.4 Garbage collection activity ............................ 141
  9.5.5 Garbage collection history .............................. 142
  9.5.6 Web application ......................................... 143
  9.5.7 EJB containers ........................................... 144
  9.5.8 Total method invocations .............................. 145
  9.5.9 Ten worst average response times ..................... 146
9.6 WSAM for scenario 2 .......................................... 147
  9.6.1 Server overview ......................................... 147
  9.6.2 Response time distribution ............................ 148
  9.6.3 Memory analysis ......................................... 149

Chapter 10. Scenario 3: Serialization ............................... 153
10.1 Overview .................................................... 154
12.3.2 Workload activity transactions ........................................... 188
12.4 TPV for scenario 5 ................................................................. 189
  12.4.1 Servlets ........................................................................... 189
12.5 OMEGAMON for scenario 5 ....................................................... 190
  12.5.1 Workload Manager (WLM) service class ............................... 190
  12.5.2 Servant regions ................................................................ 191
12.6 WebSphere Studio Application Monitor (WSAM) for scenario 5  ........................................................................... 192
  12.6.1 Server overview ............................................................... 192
  12.6.2 Memory analysis ............................................................... 193

Chapter 13. Scenario 6: Memory leak .................................................. 195
13.1 Overview .............................................................................. 196
13.2 Resource Measurement Facility (RMF) displays for scenario 6 ........ 196
13.3 RMF reports for scenario 6 ....................................................... 196
13.4 Garbage collection trace for scenario 6 ......................................... 197
13.5 TPV for scenario 6 ................................................................. 198
  13.5.1 JVM runtime .................................................................. 198
13.6 OMEGAMON for scenario 6 ....................................................... 199
13.7 WebSphere Studio Application Monitor (WSAM) for scenario 6 ........................................................................... 200
  13.7.1 Number of garbage collections versus JVM heap size ................ 200
  13.7.2 Garbage collection time versus heap size ................................. 201

Part 4. Appendixes ........................................................................... 203

Appendix A. Monitoring environment ............................................... 205
Overview ..................................................................................... 206
Infrastructure release levels .......................................................... 207
  WebSphere information ............................................................ 207
  OMEGAMON (Candle) information ............................................. 207
  WebSphere Studio Application Monitor (WSAM) (Cyanea) information ..... 208
  Tivoli Performance Viewer (TPV) ................................................. 208

Appendix B. Testing environment ..................................................... 209
Baseline setup ............................................................................. 210
WebSphere Studio Workload Simulator ............................................. 210
  WWS usage ............................................................................ 210
  Runtime Monitor .................................................................... 211
  WWS Script .......................................................................... 213
Trader Application ...................................................................... 213

Index ......................................................................................... 215

Related publications .................................................................... 217
IBM Redbooks ........................................................................... 217
Contents

Other publications .............................................................. 217
Online resources .................................................................... 218
How to get IBM Redbooks .................................................... 218
Help from IBM ..................................................................... 218

Abbreviations and acronyms .................................................. 221
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>®</th>
<th>®</th>
<th>™</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eserver®</td>
<td>Candle®</td>
<td>OS/400®</td>
</tr>
<tr>
<td>Eserver®</td>
<td>CICS®</td>
<td>Redbooks™</td>
</tr>
<tr>
<td>Redbooks (logo) ™</td>
<td>DB2®</td>
<td>RACF®</td>
</tr>
<tr>
<td>iSeries™</td>
<td>IBM®</td>
<td>RMF™</td>
</tr>
<tr>
<td>pSeries®</td>
<td>IMS™</td>
<td>Tivoli®</td>
</tr>
<tr>
<td>xSeries®</td>
<td>Lotus®</td>
<td>Virtualization Engine™</td>
</tr>
<tr>
<td>z/OS®</td>
<td>MVS™</td>
<td>WebSphere®</td>
</tr>
<tr>
<td>zSeries®</td>
<td>OMEGAMON Monitoring Agent®</td>
<td>1-2-3®</td>
</tr>
<tr>
<td>AIX®</td>
<td>OMEGAMON®</td>
<td></td>
</tr>
<tr>
<td>Candle Management Server®</td>
<td>OS/390®</td>
<td></td>
</tr>
</tbody>
</table>

The following terms are trademarks of other companies:

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

- Excel, 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, and service names may be trademarks or service marks of others.
Preface

This redbook gives a broad understanding of tools available to monitor the performance of the z/OS® WebSphere® Application Server V5.1 and the applications it is running. This redbook specifically addresses WebSphere Application Server V5.1 for z/OS and a set of monitoring tools appropriate for that release, namely; OMEGAMON V130, WSAM V3.1 and TPV for V5.1.

The team that wrote this redbook

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

Patrick Ryan is a Senior IT Architect with the IBM® Client Technology Center zSeries® eBusiness Services team and currently a Project Leader at the International Technical Support Organization, Poughkeepsie Center. He writes extensively and teaches IBM classes worldwide on all areas of the z/OS WebSphere Application Server. He has over 30 years experience with IBM mainframe operating systems, and has spent the last five years as an eBusiness consultant, dealing with the WebSphere Application Server on z/OS.

G Michael Connolly is an IT consultant at the International Technical Support Organization, Poughkeepsie Center. He has over 30 years of IBM software development experience in both distributed systems and the mainframe zSeries. He holds a BA in Humanities from Villanova University. His areas of expertise include TCP/IP Communications, UNIX® System Services, WebSphere for z/OS, and most recently the Enterprise Workload Manager (EWLM) product.

Phil Anselm is an IT Specialist with the IBM Client Technology Center zSeries eBusiness Services team. He has 21 years of experience in mainframe software configuration and performance. His areas of expertise include zSeries customization and performance tuning. He has worked extensively on CICS®, DB2®, TCP, and IMS™. He has also worked with the Poughkeepsie Center for SAP performance profiling.

Charles Neidigh is a Senior IT Architect with the IBM Client Technology Center zSeries eBusiness Services team. He has 25 years of experience in mainframe application design and development and has specialized in new technology adoption and migration. He has worked at IBM for six years. His current areas of focus include WebSphere and Java™ technologies for z/OS.
Raymond Sun is a Market Manager for Tivoli Event Automation solutions, based in Los Angeles, CA. He has 30 years of experience in Systems Management field. He holds a degree in Mathematics/Computer Science from the University of California at Los Angeles. His areas of expertise include MVS™, JES2, Automation, Performance Management, WebSphere MQ, WebSphere MQ Integrator, WebSphere Application Server, and OMEGAMON®. He has taught extensively on mainframe performance management and mainframe automation.

Theresa Tai is a Senior Software Engineer at the zSeries New Technology Center in Poughkeepsie, New York. She started out as a system programmer in JES3 and JES2 Design, Development, and Service. For the past six years, she has been supporting the WebSphere Application Server on z/OS, Java, and Portal Server. She holds a degree in Computer Science from CUNY. Her areas of expertise include WebSphere on z/OS and Portal server on zSeries. She has contributed extensively to several redbooks on the subject of WebSphere Application Server and WebSphere Portal Server.

Michael Weintraub is a freelance IT consultant based in Berlin, Germany. He has 32 years of experience in monitoring, automation, performance and capacity management, including over 20 years in different international roles with Boole & Babbage and later BMC Software. He holds a degree in mathematics from the Warsaw University, Poland. His areas of expertise include mainframe and client/server performance, infrastructure design, business process automation, middleware, and many others. He is currently deputy chairman of the German Guide/Share Europe Linux® Working Group and is an active member of other professional organizations, such as ceCMG. Known for his lateral thinking, Michael is also often a featured speaker at many user conferences.

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

Richard M Conway (for the 100th plus time)
International Technical Support Organization, Poughkeepsie Center

Aris Katsoulis
IBM US

Barry Lamkin
IBM US

Become a published author

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

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

► Use the online Contact us review redbook form found at:

ibm.com/redbooks

► Send your comments in an email to:

redbook@us.ibm.com

► Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYJ Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400
Part 1

Theory and methodology

In this part, we discuss our theory of what performance monitoring means and also the methodology that we recommend for monitoring performance.
Theory of performance monitoring

In this chapter, we discuss the theory underlying performance monitoring and management.
1.1 Introduction

This redbook provides an overview of how to use the monitoring tools appropriate for the WebSphere Application Server V5.1 for z/OS. In order to do this, we provide a number of scenarios, and use the version of the tools available for WebSphere Application Server V5.1 to monitor performance and compare it to a set of baseline runs that we performed.

**Important:** For WebSphere Application Server V6 or higher we strongly suggest that you review the performance monitoring publications that apply to those releases. The monitoring tools for these later releases may differ significantly from those we describe in this redbook. Regardless of your level of WebSphere Application Server, we believe the approaches and techniques we describe in this Redbook for monitoring workloads will still be of benefit.

1.2 Overview

In order to start, we consider some basic questions such as what is performance, why is it monitored, and how?

The first step in understanding performance is defining what is meant by performance. A simple definition of application performance is how quickly a transaction carries out a specified task. However, this is a very vague and subjective definition, and must be refined.

In a constantly changing production environment, the ability to predict and track transaction behavior is critical to an organization’s ability to proactively identify problems before a serious failure occurs. Although the ultimate goal is to monitor performance in the production environment, an effective monitoring process involves nearly all stages of the application life cycle. This includes early design decisions through development and testing, and finally to deployment and maintenance.

Performance monitoring continues to become more important as the requirements for WebSphere-based applications become more stringent, particularly in terms of availability and capacity. The focus of this book is to establish a basic performance monitoring model and provide the reader with a number of methods and tools that aid in the task of monitoring performance.

For realistic performance monitoring, the definition of how the system should perform must meet the following three criteria:

- First, the definition must be *quantitative*. That is, it must facilitate objective measurement.
Secondly, it must be consistent. Long-term performance monitoring requires comparison of measurements over extended periods of time, although execution environments, server configurations, or applications may change.

Finally, it must be realistic. The performance model must account for the real constraints of the application and not simply a desired goal. Not all transactions perform the same amount of work and, therefore, they will not perform in the same way.

1.3 Performance metrics

This section discusses various performance metrics, components, and expectations that are used to define a quantitative, consistent, and realistic performance monitoring model. The following are some of the typical performance metrics used in monitoring and how they are used in the scenarios of this book.

Transactions
The first metric is that of a transaction: a single, complete unit of work that is the unit that all of the other metrics typically measure. In the broadest sense, a transaction is a complete business task from the point of view of the ultimate consumer, either a human user at a browser or an automated process, which consists of a single request and response indicating the completion of the request.
This logical business transaction (LBT) or end-to-end transaction typically encompasses multiple components to complete a request. These components comprise a browser, an HTTP server, one or more application servers, back-end servers, database servers, as well as internal networking components and the external Internet. Figure 1-1 illustrates an end-to-end transaction infrastructure.

![End-to-end transaction infrastructure](image)

**End-to-end transaction monitoring**

As transactions become more complex, establishing transactional boundaries and measuring performance has also become more complex. End-to-end transaction monitoring and measurement is becoming an area of great interest to a lot of organizations.
A discussion of end-to-end transaction monitoring and measurement is beyond the scope of this book. This book deals with only a subset of the full LBT, those that enter a single WebSphere Application Server for z/OS server as either a Hypertext Transfer Protocol (HTTP) request or an Internet Inter-ORB Protocol (IIOP) request, see Figure 1-2.

![Server Transaction Infrastructure](image)

In some cases, it may be necessary to establish the ratio between the LBT and individual server transactions, as some applications may make several calls to an application server for a single business transaction. However, in the scenarios used here, we use a simple one-to-one structure.

**Response time**

The most common metric for measuring performance is response time. Response time is the time (typically in ms) required to complete a transaction. Figure 1-3 illustrates the response time breakdown.

For a WebSphere Application Server for z/OS transaction, the response time consists of two main parts:

- The time the transaction waits for an execution thread, referred to as the *queued time* or *wait time*.
- The time taken to perform the transaction or *execution time*. 
Execution time consists of the actual processing time, back-end wait time, and any delay time. Back-end wait time is the time spent waiting for a response to a call to a back-end process such as a call to a database server, another transaction server such as a Customer Information Control System (CICS) or an Information Management System (IMS), or to a remote Enterprise JavaBeans™ (EJB™) that is deployed on another server.

Delay time accounts for the amount of time when a transaction cannot process due to the constraint of a system resource. Delays can be caused by a number of different system resources. Most often, delays are caused by CPU availability, real memory availability, or input/output (I/O).

![Diagram of Total Transaction Response Time]

Figure 1-3  Response time breakdown

The theoretical minimum response time for a transaction refers to the lowest possible response time, given that there are no system constraints, that is, no queue time and no delay. Therefore, the minimum response time includes only the actual processing time plus any back-end wait time.

**Throughput**

In the z/OS environment, the individual transaction response time is usually less important compared to the total number of transactions that can be run in a specific period of time. This is referred to as *throughput* and is typically measured in transactions per second (tps).

The importance of throughput is based on one of the key advantages of the z/OS environment, that is, its ability to manage multiple, diverse workloads while maintaining acceptable response times. Throughput takes into consideration
multiple execution threads, client delivery rates, and the sharing of resources between threads.

Getting the optimal throughput for a workload requires establishing a balance between the threads, resources, application characteristics, and business requirements. The theoretical maximum throughput assumes no constraints or contention. It is the minimum transaction response time multiplied by the number of execution threads.

**Number of clients, delivery rate, and server utilization**

The number of clients, delivery rates, and server utilization are measures of the load of transactions on a server and can represent the actual, expected, or maximum load.

**Number of clients**

The number of clients is the count of simultaneous users making requests to the server. However, because of the nature of Web applications, the number of simultaneous users is not directly translated to the number of simultaneous transactions that are executing. A user waits only for a small portion of the time for a response from the application. The user spends the rest of the time reading, interpreting, or entering data which is referred to as the **think time**.

**Delivery rate**

More important to monitoring performance on the WebSphere Application Server for z/OS is the delivery rate (in requests per second), that is the rate at which new requests arrive at the server. Delivery rate is roughly the number of clients divided by the sum of think time, average transaction response time, and any network delay.

**Server utilization**

Finally, server utilization is the ratio of the delivery rate and maximum throughput and is an indication of how fully a server is being used. Reaching 100% utilization is impractical for most servers, since this assumes no delays, no contention, and a perfectly steady stream of incoming requests. Only actual experience with the server and application workload can determine the saturation point for server utilization.

**System resource metrics**

The two key system resources that affect Java applications are central processing unit (CPU) and real memory. CPU utilization can be stated in either CPU time (typically in ms) or as a percentage of the overall CPU capacity.
**CPU utilization**

In performance monitoring, CPU percentage is less useful because it can be very dynamic and difficult to compare one time period to another or compare one platform to another. Although CPU time is more consistent, this too can be difficult to compare across different hardware platforms.

**Memory utilization**

Several metrics are used in monitoring memory utilization. From a system standpoint, paging rate indicates the number of times (usually in seconds) that a required segment of memory has to be read back into real memory from auxiliary memory. A WebSphere Application Server uses a significant amount of real memory compared to other z/OS hosted transaction servers. System paging can have an impact on the transaction response time.

From a server standpoint, the key memory metrics are:

- Average and maximum heap size indicating the amount of memory available to the Java virtual machine (JVM™).
- Garbage collection intervals indicating the time period between the garbage collections.
- Garbage collection time indicating the amount of time required to complete a garbage collection cycle.

The garbage collection percentage is calculated as the garbage collection time divided by the garbage collection interval multiplied by 100. The desired garbage collection percentage is between 2% and 5%.

**Workload Manager metrics**

Two additional metrics are available from the Workload Manager (WLM) that indicate the performance of a service class against the established goal. These are the Performance Index (PI) and Workflow Percentage.

As a single, simple value, the PI is a key metric for triggering review of the server performance. The PI is a value of 1 when the overall workload performs at or near the goal. It is less than 1 when the overall workload outperforms the goal, and greater than 1 when it does not meet the goal.

The Workflow Percentage shows how well the service class performs by indicating how close the actual response time is to the maximum response time. A value of 100% indicates that the current average response time has no queue time or delays.
1.4 Components impacting performance

In order to effectively monitor performance, it is important to understand the impact of the various components involved. The overall performance is the result of the interaction of three main components: the application, the WebSphere Application Server in which it runs, and the z/OS operating system that provides the CPU, memory, I/O, and other system resources required by the application.

1.4.1 Application performance factors

Nothing has a greater impact on application performance than the application itself. How the application is designed and coded, the amount of work within a transaction, the complexity of a transaction, all of these factors impact the performance characteristics of the application.

Application types

Breaking applications into specific application or transaction types, based on specific characteristics, helps to match an application to the application server that it is expected to run on. Categorizing applications also helps in determining the desired characteristics of a new application by using the experience of similar existing applications to predict behavior.

One of the most common characteristics used for categorizing applications is the application structure or pattern. Examples of application patterns include:

- A simple Web application which consists of a few servlets and JavaServer™ Pages™ (JSPs).
- A simple Java 2 Platform, Enterprise Edition (J2EE™) application, which consists of servlets, JSPs, and one or more local EJBs.
- A complex J2EE application, which uses calls to remote EJBs or other external processes such as CICS or IMS transactions, or extends a global transaction across multiple transaction managers.
- A J2EE front-end application, which uses servlets, JSPs, and EJBs to access existing back-end CICS or IMS transaction where the majority of the processing is done in the back-end process.
- A Web service application, which uses the Web service interface to isolate the transaction.
- An asynchronous transaction, which uses Java Message Service (JMS) or other messaging methods to initiate the transaction, but the initiating client does not wait for completion.
These are just a sample of the possible patterns. New patterns are constantly being established as the J2EE specification evolves. In addition to the application pattern, other factors can be used in defining application types including:

- Expected volumes: High volume transactions versus infrequently used transactions.
- Business priority: Some applications or transactions may be more important to the organization than others and, therefore, require higher performance characteristics.

**Application design and coding**

Taking some important steps in application design and coding can ensure the best possible performance of the transactions. These include:

- Design to meet nonfunctional requirements: Understanding the performance, availability, scalability, security, and other nonfunctional criteria of an application is critical to the proper design and to establishing a proper expectation for performance. One key area is understanding the true business requirements for transaction volumes and response times.

  A number of design decisions are dependent on these requirements, particularly the amount of work that can be completed within the transaction and the types of connectors or subsystems that can be used. In addition, other nonfunctional requirements such as availability, security, or reusability have an impact on the performance. Over-engineering applications to meet unnecessary requirements may cause significant performance and scalability repercussions.

- Use of established design patterns and frameworks: Established patterns and frameworks can make the development process easier and more consistent. It can also make assessing the performance characteristics of an application easier. By comparing applications that are functionally and structurally similar, developers can quickly see possible performance problems early in the development process.

- Strict adherence to J2EE specifications: The J2EE specification was developed to allow for multi-threading and for multiple applications to share servers. Both of these can be critical to deployment in a shared environment such as WebSphere Application Server on z/OS. Improper use of threading, serialization, or static variables can have a severe impact on this capability.

- Following coding best practices: Several sources of information on best practices are available and can help developers to avoid many common mistakes, the majority of which show up in performance problems late in the project.
Performance testing and application profiling: Testing and profiling prior to deployment is another means to compare performance in the production environment.

1.4.2 WebSphere Application Server for z/OS

The WebSphere Application Server for z/OS has several features to help customize the application server to improve performance.

First, it is important to ensure that the servers have been properly installed and tuned. The Tuning section of the WebSphere Application Server Infocenter contains basic tuning tips for the Object Request Broker (ORB) service, Extensible Markup Language (XML) parser, dynamic cache service, EJB container, session management, and data sources.

Refer to the WebSphere Application Server Infocenter at:
http://publib.boulder.ibm.com/infocenter/ws51help/index.jsp

After creating a basic server, the three primary areas of performance impact to be concerned with are defining the number of execution threads, allocating memory, and determining connections to back-end subsystems.

Server threads

Each transaction in a WebSphere Application Server for z/OS server is run in its own thread or enclave. The number of server threads determines how many concurrent transactions are running at a specific time.

To obtain optimum performance, it is important to balance the number of execution threads with the characteristics of the transactions and CPU availability. Make an initial calculation of the threads required for a given workload by multiplying the expected delivery rate by the average transaction execution time.

The number of execution threads available within a logical server can be affected in the following three ways.

WORKLOAD_PROFILE parameter

First, the WORKLOAD_PROFILE parameter establishes the number of threads in a single servant region. You can find The WORKLOAD_PROFILE parameter in the administrative console at Servers → Application Servers → server name → ORB service → Advanced Settings. The values for this parameter are:

- IOBOUND: This is the default value and the number of threads is three times the number of CPUs available, with a minimum of five and a maximum of 30. This setting is appropriate for most types of workloads.
Monitoring WebSphere Application Performance on z/OS

- **CPUBOUND**: Provides for a number of threads equal to the number of CPUs available minus one, with a minimum of three. This setting is for applications that are very CPU intensive.

- **LONGWAIT**: Provides a fixed number of 40 threads. This setting is appropriate for applications where the majority of processing is done in a back-end subsystem, such as a transaction that is only a front-end for a CICS or an IMS transaction.

- **ISOLATE**: Provides only one thread. This setting is only for applications that must run single threaded. Significant scalability problems can arise from using this value, therefore, take care when using this value.

**Multiple servant regions**

The second way to alter the number of execution threads is by allowing multiple servant regions instances within a single server control region. To do this, set the Minimum Number of Instances, Maximum Number of Instances, and Multiple Instances Enabled parameters, which you can find in the administrative console at Servers → Application Servers → server_name → Server Instance.

The user can add server instances explicitly at startup by increasing the value of the Minimum Number of Instances. WLM will dynamically start another server instance when it detects the need for more threads if the Maximum Number of Instances is greater than the Minimum Number of Instances.

This provides a small buffer for occasional spikes in incoming requests. However, if the server instances are regularly started dynamically, increase the Minimum Number of Instances so that the overhead of starting a server instance is done at startup time.

Also, when the Multiple Instances Allowed flag is set, always define the Maximum Number of Instances. We recommend that the value should **not** be more than one or two greater than the minimum. This keeps the WLM from starting too many server instances and potentially creating a memory or other system resource shortage.

**Note**: Adding additional server instances in this fashion is only useful when CPU resources are available to service the threads. If CPU delays are already present in a logical partition (LPAR), adding additional threads may reduce queue time. However, CPU delay time will become more and, therefore, the overall response time will not improve.

**Adding cloned servers**

Finally, the user can change the number of execution threads by adding one or more cloned servers to a server cluster. This facility is primarily used to add
availability and failover capabilities to the server. It also adds server instances, providing additional execution threads, and it adds the additional CPU and memory of another LPAR.

**Memory allocation**
The JVM heap size is another area where the user can customize a server to the workload. Here again, optimum performance requires a balance between the memory available in each server, the memory usage characteristics of the applications, and the real memory available on the system.

You can find the prearrangers for modifying the JVM heap size in the administrative console at Servers → Application Servers → `server_name` → Process Definition → Java Virtual Machine.

**Connectors**
Connecting to back-end systems generally involves a great deal of communication and movement of data, which can become a bottleneck within the server. Each resource can have multiple connector types and unique tuning or connection pooling requirements.

**WebSphere Performance Monitoring Interface**
The WebSphere Performance Monitoring Interface (PMI) significantly increases the availability of performance data. The PMI service collects both the server and application data that can be accessed through different interfaces. It can collect data at different monitoring levels, and allows for the monitoring levels to be modified dynamically.

1.4.3 **z/OS environment**

**Work Load Manager**
The Work Load Manager (WLM) affects the WebSphere Application Server for z/OS in two ways. First, the WLM manages transaction response times based on predetermined response time goals, workload mix, and system resource availability.

Secondly, the WLM has the capability to dynamically increase and later decrease the number of servant regions running for a given server, based on the Minimum Number of Instances and Maximum Number of Instances values defined in the administrative console. This allows the WLM to respond to fluctuations in request volume.
**UNIX System Services**

WebSphere is a UNIX System Services (USS) application, that is, it runs on the UNIX environment under z/OS. WebSphere uses a large number of USS threads, files, processes, and sockets, and, therefore, the basic configuration of USS can impact WebSphere performance.

USS also uses a hierarchical file system (HFS) similar to that implemented on other UNIX platforms. The user must properly use and tune this system to ensure good performance. Basic tuning guidelines for USS are available in the WebSphere infocenter.

**Transmission Control Protocol/Internet Protocol**

Like USS, WebSphere interacts very closely with Transmission Control Protocol/Internet Protocol (TCP/IP), which controls the network connections. Tune the basic TCP/IP configuration according to the guidelines available in the WebSphere infocenter.

### 1.5 Setting expectations

Performance measuring and monitoring can gauge the level at which a given transaction or an entire workload is performing, but when is it performing well enough? Setting a performance expectation or target is necessary to evaluate the metrics collected. Establishing the expectation comprises a variety of sources including:

- **Business volumes and response requirements:** This involves defining the performance level that the application is expected to meet from a business perspective. Volume, in particular, significantly influences the efficiency of a transaction. For example, a transaction that is expected to run hundreds or even thousands of times a minute must be much more efficient than one that runs hundred times a day.

- **Application profiling:** Profiling an application, that is, breaking the execution of a single transaction down to its smallest measurable parts, shows where the majority of the processing is taking place. This can uncover inefficiencies in particular components or methods.

- **Load testing:** By testing the application under load using a workload simulation tool such as WebSphere Studio Workload Simulator, you can evaluate the application behavior at different volume levels and assess the scalability of the application and server.

- **Monitoring:** Monitoring provides a baseline that can be used to compare performance over time. This helps to identify performance changes in existing
applications and servers caused by new deployments or software upgrades, or changes in the system environment.

- **Experience:** In course of time, the characteristics and capabilities of servers and applications become better known and can be extrapolated to new servers and applications of a similar function or structure.

When defining the performance expectation be careful to avoid requirements that are arbitrary or unrealistic. Common mistakes include:

- **Response times goals that are not based on the amount or type of work being done or are not required by overall volume.** For example, stating that a transaction must complete in 100 ms is arbitrary.

- **Comparing performance across different platforms or environments, particularly at an individual transaction level.** Comparing the response time of a single transaction running on a distributed platform to one on a z/OS platform is not a valid indicator of how a transaction will perform under load in a production environment. Similarly, it is not valid to compare a J2EE transaction running on WebSphere to a transaction running on CICS or IMS.
Methodology of performance monitoring

In this chapter, we discuss the use of a performance monitoring methodology and how the monitoring tools fits into the methodology tasks.
2.1 Overview

A performance monitoring methodology, like any other methodology, defines the basic processes and tasks, which are used to establish a consistent, proactive monitoring environment.

The simple monitoring cycle in Figure 2-1 shows a five-step iterative process. The steps in this cycle can be performed by the same person or group. In larger organizations, each step can be performed by different departments or groups and require a much more formal interface between steps.

![Figure 2-1 Performance monitoring methodology](image-url)
2.2 Performance monitoring cycle

This section describes the basic function of each of the steps of the performance monitoring methodology and the triggering of subsequent steps.

2.2.1 Initial installation

A complete installation process will most likely only be performed once, the first time the WebSphere Application Server for z/OS is installed at a site. The purpose of this step is to evaluate all of the performance-related parameters involved in the cell, node, and server configuration and to ensure that the initial setup is correct.

It is also important to know and understand the values in these settings, particularly since many of them may be default at installation time. The default values are usually appropriate for first-time installations when little is known about the applications or the environment.

Modifications should be based on specific application knowledge and experience. The WebSphere Infocenter provides guidance for performance tuning and the various configuration settings.

Another point at which this step may be performed is during a major version upgrade. Configuration settings or default values may change between releases or settings added or removed. For example, in the migration from WebSphere Application Server V4.0 to WebSphere Application Server V5, the default threading model, as defined in the WORKLOAD_PROFILE parameter, was changed. This could have significant impact on an existing application’s performance.

Always consult the migration information and the performance tuning information in the WebSphere Infocenter for each new release and re-evaluate the settings being used. A new installation or major upgrade should always trigger a new baseline step. Performance enhancements are part of almost all upgrades and you should quickly identify any configuration issues. Expect changes in application performance behavior and establish a new baseline for monitoring.

2.2.2 Setting a baseline

Establishing a baseline value for metrics is the key to ongoing performance monitoring. Specific numbers, for example, response time or CPU utilization, have much less value if there is nothing to compare them with and determining if and where a performance problem exists is much more difficult. The baseline establishes what is normal for a server. Capture the baseline numbers for each server in a WebSphere cell.
In a production environment, this may require intense monitoring for a short period of time to determine what the normal levels are. Identify a period of time that represents the peak activity for a normal business cycle, such as a one-hour or two-hour period for each business day, and take measurements over several periods. Save these measurements as the baseline.

Performance testing in a non-production WebSphere server or cell may also produce a baseline information. However, unless the testing can accurately replicate transaction volumes, system capacity, workload mix, and other variables that impact the production environment, the numbers may not be directly applicable to the production environment. Nevertheless, establishing and storing baselines from non-production performance testing can still be very valuable in comparing various releases of the application code.

**Setting the WLM goal**

When the normal server behavior is known, evaluate the WLM goal to determine if it is appropriate for the behavior. In our test to establish a baseline scenario, we started with a very relaxed goal of 90% at 1 second response time. After running a couple of cycles, we used the Resource Measurement Facility (RMF™) workload reports to assess the workload distribution and reset our goal to 94% at .700 seconds.

Our intent was to get the performance indicator as close to 1 as possible for our normal processing. Setting a goal that is too tight will cause servers to fail to meet the goal on a regular basis. Setting a goal that is too loose will cause servers to almost always meet the goal, and will hide peaks, changes in performance, and other trends until they become serious problems.

**Setting performance triggers**

Performance triggers are limits established for specific metrics, at which point an evaluation of the metric is necessary. Multiple triggers can be used and can have different levels of importance. Triggers can also be tied to multiple periods in order to avoid unnecessary concern for momentary peaks. It is important to identify the metric being measured, the limit or limits that cause a trigger to occur, the importance level of each limit, and the notification process each time a trigger occurs.

Examples of such triggers are:

- Performance index is greater than 1 for more than 5 measuring periods
- Performance Index is greater than 4 for any period
- CPU utilization is greater than 98%
- Response time is greater than a specific value
- Queued time is greater than a specific value
- Number of incoming requests is greater than a specific value
Some of the interactive tools that are discussed later provide customizable limits and triggering processes, including automated notification processes. Even when not using these tools, establish some form of triggering process to identify changes in performance behavior.

### 2.2.3 Monitoring

The heart of the cycle is the monitoring step, which is a continuous process to measure and compare performance metrics at predetermined intervals. The basic tasks of monitoring are:

- Collecting periodic performance data
- Comparing to pre-set limits and triggering a performance event if a limit is exceeded
- Notifying the appropriate persons or groups of the triggering event

The ways in which monitoring can take place is categorized as follows:

- Real-time vs. historical
- Manual vs. automatic
- Normal vs. diagnostic

Many organizations will choose to use a mixture of different monitoring methods and tools to develop a full performance picture.

**Real-time monitoring versus historical monitoring**

In this section, we describe the real-time monitoring and the historic monitoring processes.

**Real-time monitoring**

Real-time monitoring uses interactive tools to view performance data in very short time intervals, measured in minutes or seconds. This tends to be the most invasive form of monitoring because of the amount of performance data collected at the server level. Therefore, take care not to over instrument a server, thereby causing a performance problem rather than monitoring for one.

The advantage of real-time monitoring is the immediate feedback, which gives the user the ability to react quickly to any triggering event and to see momentary peaks. In addition, many of the interactive monitoring tools have the ability to dynamically alter the level of performance data being obtained. This is very useful when monitoring to diagnose a specific problem or a specific server. This capability allows you to start with a higher level and then narrow in on the problem.
**Historical monitoring**

Historical monitoring involves storing performance data, which is retrieved and analyzed at a later point in time. One of the best examples of this is the RMF workload analysis report, which we discuss later.

There are several advantages to this type of monitoring. First, data can be collected and stored in a way that allows for easy comparison of different periods or intervals. Secondly, this monitoring typically covers a much larger time interval. This can smooth out momentary peaks and makes for easier identification of long-term trends in performance. Finally, this tends to be a less invasive process, causing little or no impact on the monitored server.

However, the disadvantage is that issues identified are not known until well after the event has taken place. Gathering additional information about a triggering event then becomes more difficult. Another consideration is that historical monitoring can accumulate a very large amount of data over time. A process to limit the data stored for long periods of time is necessary to ensure that storage and retrieval of meaningful data is feasible.

**Manual versus automatic**

Manual monitoring requires a knowledgeable person to drive the monitoring process, to compare data, to identify a triggering event, and to notify the appropriate areas of the event. This can be a very labor-intensive process, which is prone to the usual human mistakes and inconsistencies. Manual monitoring is best only for short-term, diagnostic types of monitoring tasks.

However, some tools have a limit to the amount of automation provided and attaining a fully automated process may require some inventive processing on the part of the users. Tools that offer more automatic features free resources for more diagnostic and analytic tasks. The more automation that is added to the basic monitoring tasks, the more consistent the data will be, and the user can compare more data. But a greater dependency is placed on setting trigger levels properly.

**Normal versus diagnostic**

Normal monitoring refers to the day-to-day monitoring of stable, well-tuned servers. It is the lowest level of monitoring necessary to catch any basic changes in behavior. Several approaches can be used to obtain this level such as using longer intervals for data collection, collecting fewer metrics, or by using historical monitoring as opposed to real-time monitoring.

Diagnostic monitoring is a more short-term task that is focused on a specific problem area and tends to collect more performance data and be more invasive. In a full production cell with multiple servers, different servers may require
different levels of monitoring based on things such as new application
deployments, different volume levels or importance of applications, or specific
problems with an application or server. Tools that can customize or dynamically
alter monitoring levels make it easier to set lowest level of monitoring necessary
for each server.

### 2.2.4 Evaluating

After the monitoring step has identified a triggering event, start an evaluation
process. This is to determine if the event is a real problem, just an anomaly, or an
unusual peak of some kind that has resolved itself and requires no further action.
The basic tasks of the evaluation process may include:

- Assessing initial triggers to determine which limit or combination of limits were
  exceeded and what other key performance metrics were at the time of the
  event.
- Additional diagnostic monitoring to obtain more information about the specific
  application, server, or z/OS system performance.
- Additional testing to recreate the event in a non-production environment.
- Logging event information and reviewing logs for past events that may
  indicate some form of pattern.

When the event has been identified as a problem, that is, additional action is
required, the goal of the evaluation step is to isolate the root cause of the
problem. In many cases, this becomes a process of elimination. For this purpose,
some form of sequential process or checklist is useful to eliminate any common
or well-known problems and to isolate the problem into a specific area.

You can also use combinations of metrics to narrow the focus of the evaluation.
For instance, if an event is triggered due to high transaction response times and,
at the same time, the number of completed transactions is significantly higher
than normal, the focus of the evaluation will be on the cause of the increase in
volume, and whether it represents an unusual single occurrence, a previously
unknown periodic peak, or a general trend towards higher volume.

### 2.2.5 Modifying

The final step of the cycle is to modify one or more of the basic components to
resolve the performance problem, or to adjust the performance model to match
the new understanding of the production environment. These changes include:

- Modifying and redeploying the application
- Modifying the server configuration
- Modifying the z/OS environment or hardware
After all modifications are complete, a new baseline step is initiated and the cycle is restarted.

2.2.6 Summary

In summary, by using a well-established methodology supported by the proper tools, performance monitoring can be an effective and proactive process instead of a casual and haphazard occurrence.
Monitoring tools

In this part, we describe some of the actual tools that you can use to monitor performance. The tools are grouped into three categories:

- Basic tools
- WebSphere Application Server tools
- Additional tools
Chapter 3. Basic tools

In this chapter, we describe and discuss the following basic tools that are available on the z/OS WebSphere Application Server environment:

- System Management Facility (SMF) records
- Resource Measurement Facility (RMF)
- JAVA garbage collection (GC) verbose trace
- Workload Manager (WLM) policies
3.1 System Management Facilities records

System Management Facilities (SMF) is a good source of information for system and subsystem performance data. It can record information from most system components, including Hypertext Transfer Protocol (HTTP) and WebSphere Application Servers.

For performance analysis and depending on your software environment, the following SMF records provide useful information:

- Record type 70 - 79: RMF records, especially record type 70 for processor activity and type 72 for workload activity
- Record type 88: system logger activity
- Record type 92: UNIX System Services (USS) hierarchical file system (HFS) information
- Record type 103: HTTP Server information
- Record type 100 - 102: DB2 statistics, accounting, performance
- Record type 110: Customer Information Control System (CICS) TS statistics
- Record type 115, 116: WebSphere MQ statistics
- Record type 120: WebSphere Application Server information. The administrative console allows you to switch recording ON or OFF selectively for activity records or interval records.

For more information on the use of SMF records for DB2, CICS, WebSphere MQ or TCP/IP, refer to the documentation for each subsystem.

3.1.1 The z/OS SMF Summary Viewer

WebSphere Application Server V5 for z/OS Performance Summary Report for SMF records

This version of the SMF browser is an extension of the program distributed from the WebSphere V5 for z/OS download site at:


This version adds a summary report showing activity for each Java 2 Platform, Enterprise Edition (J2EE) server instance, bean, and method from the SMF type 120 records. It is a Java utility, therefore, it has to run in a Java Virtual Machine (JVM) in the UNIX environment.
This tool may be updated in the future. For the latest version, see the latest copy of this document number PRS752 in the “Presentations & Tools” category of Techdocs at:

http://www.ibm.com/support/techdocs/

This version of the SMF browser adds a summary report, which is much more easy to read than the standard report. It shows the activity for each Java 2 Platform, Enterprise Edition (J2EE) server instance, bean, and method from the SMF type 120 records.

3.2 Resource Measurement Facility

Resource Measurement Facility (RMF) is a performance measurement and management product that is shipped with every release of z/OS. The base product collects performance data for z/OS and sysplex environments, and provides a variety of means to display, analyze, and compare this data to optimally configure a system.

The four RMF components discussed in this section include:

- RMF Monitor III interactive displays
- RMF PM, a Java-based workstation interface
- RMF Monitor I reports
- RMF Spreadsheet Reporter, a data interface to common spreadsheet tools

For more information about all of the RMF tools, refer to the RMF home page at:


3.2.1 RMF Monitor III displays

RMF Monitor III provides online, real-time snapshot monitoring and short-term performance analysis. Much of the same performance information available on the RMF postprocessor reports are available using Monitor III.

Of the many RMF Monitor III displays available, we focus on the following for monitoring the WebSphere Application Server for z/OS:

- For monitoring service class performance compared to the Workload Manager (WLM) goal, the equivalent to the RMF Workload Activity report:
  - Sysplex summary display (SYSSUM)
  - Response time distribution display (SYSRTD)
  - Work Manager delays display (SYSWKM)
For monitoring the overall system performance, the equivalent to the central processing unit (CPU) and summary reports:
- System information display (SYSINFO)
- CPC capacity display (CPC)

Other displays are also available to further investigate CPU and storage delays, and other system-wide resource utilization.

### 3.2.2 RMF PM

An alternative to the 3270-based RMF Monitor III displays, RMF PM is a workstation-based, graphical user interface (GUI) application that presents the same RMF data in a graphical fashion. The interface allows the user to build customized desktops (see Figure 3-1) focusing on the WASPM service class.

![RMF PM desktop](image)

*Figure 3-1 RMF PM desktop*
3.2.3 RMF Monitor I reports

RMF historical reporting or postprocessor allows for the creation of a number of reports from historical SMF data. The reports that are of most significance in the monitoring of WebSphere Application Server for z/OS are:

- Workload activity reports
- Summary and CPU reports

Dumping SMF records

Creating RMF reports is a multi-step process. First, extract SMF data to a sequential data set using the SMF program IFASMFDP, see Example 3-1. During this process, you can filter the SMF data to include only the type 70 through 78 records.

Example 3-1 Job control language (JCL) for dumping SMF records

```plaintext
//SMFDUMP EXEC PGM=IFASMFDP,
//IDD1 DD DISP=SHR, DSN=<input_smfdata_system1>
//IDD2 DD DISP=SHR, DSN=<input_smfdata_system2>
//SMFDATA DD DISP=(NEW,CATLG), SPACE=(CYL,(10,10),RLSE),
//                  UNIT=SYDA, DCB=(RECFM=VBS, LRECL=32760, BLKSIZE=0),
//                  DSN=FRANCK.SMF.D06T1700

//SYSIN DD *
  INDD(IDD1, OPTIONS(DUMP))
  INDD(IDD2, OPTIONS(DUMP))
  OUTDD(SMFDATA, TYPE(70:78))
```

Postprocessing

Next, sort the SMF records and create the RMF reports, as shown in Example 3-2.

Example 3-2 JCL for running the postprocessor

```plaintext
//RMFSORT EXEC PGM=SORT, REGION=0M
****** SORTIN DATA SETS FOLLOWING HERE ***********
//SORTIN DD DISP=SHR,
  DSN=FRANCK.SMF.D06T1700
//SORTOUT DD DISP=(NEW,PASS), DSN=&SORTOUT, UNIT=SYSALLDA,
  SPACE=(CYL,(50,50)), DCB=*.RMFSORT.SORTIN
//SORTWK01 DD DISP=(NEW,DELETE),
  DSN=&WK1, UNIT=SYSALLDA, SPACE=(CYL,(50,50)),
  UNIT=SYSALLDA, SPACE=(CYL,(50,50))
//SYSPRINT DD SYSOUT=* 
//SYSOUT DD SYSOUT=* 
//SYSIN DD *
  SORT FIELDS=(11,4,CH,A,7,4,CH,A),EQUALS
```
MODS E15=(ERBPPSRT,500),E35=(ERBPPSRT,500)
//POST1 EXEC PGM=ERBRMFPP
//MFPINPUT DD DSN=*.RMFSORT.SORTOUT,DISP=(OLD,PASS)
//*
//SYSIN DD *
RTOD(0000,2400)
STOD(0000,2400)
REPORTS (CPU)
SUMMARY (INT)
SYSOUT(T)
//POST2 EXEC PGM=ERBRMFPP
//MFPINPUT DD DSN=*.RMFSORT.SORTOUT,DISP=(OLD,PASS)
//*
//SYSIN DD *
RTOD(0000,2400)
STOD(0000,2400)
SYSRPTS (WLMGL(RCLASS(WAS*,OTHER,SYS*)))
SYSRPTS (WLMGL(POLICY,SCP(WAS*)))
SYSOUT(T)

Workload activity report

Summary and CPU activity reports
These two reports, generated by the SUMMARY (INT) and REPORTS (CPU) control statements provide additional information about the overall system for a specified interval. The summary report provides CPU busy, SWAP RATE, and DEMAND PAGING statistics that are useful for identifying system constraints that may influence transaction behavior. The CPU activity report gives further detail about the overall CPU and memory utilization.

3.2.4 RMF Spreadsheet Reporter (RMFPP)

RMF Spreadsheet Reporter is a graphical, workstation-based extension to the RMF postprocessor reports. Spreadsheet Reporter provides RMF data that can be imported into Microsoft® Excel® or Lotus® 1-2-3®, and along with provided macros, to build graphical reports for performance monitoring and trend analysis.
3.3 **JAVA garbage collection verbose trace**

WebSphere run applications in a Java virtual machine (JVM). When this JVM fails to allocate memory due to a shortage of Java heap, it starts garbage collection (GC). To identify if the JVM heap size is large enough or if there is a memory leak, you can collect a verbose GC trace.

To turn on a verbose GC trace in WebSphere Application Server, use the administrative console or edit tool to set JVM_ENABLE_VERBOSE_GC=1 in the current.env file. The verbose GC trace is sent to sysout of the server region job.

### 3.3.1 Verbose trace output

Example 3-3 illustrates a GC trace output.

*Example 3-3  GC trace output*

```
JVMST080: -verbose:gc flag is set
JVMST082: -verbose:gc output will be written to stderr
JVMST080: -verbose:gc flag is set
JVMST082: -verbose:gc output will be written to stderr
<GC[0]: Expanded System Heap by 65536 bytes
<GC[0]: Expanded System Heap by 65536 bytes
<GC[0]: Expanded System Heap by 65536 bytes
<AF[1]: Allocation Failure. need 16400 bytes, 0 ms since last AF>
  <AF[1]: managing allocation failure, action=1 (14168/131004928) (3145728/3145728)>
  <GC(1): GC cycle started Thu Dec  5 15:49:25 2002
  <GC(1): freed 115391688 bytes, **88% free** (118551584/134150656), in 93 ms>
    <GC(1): mark: 77 ms, sweep: 16 ms, compact: 0 ms>
    <GC(1): refs: soft 0 (age >= 32), weak 40, final 414, phantom 0>
  <AF[1]: completed in 93 ms>
<AF[2]: Allocation Failure. need 40 bytes, 59144 ms since last AF>
<AF[2]: managing allocation failure, action=1 (0/131004928) (3145728/3145728)>
<GC(2): GC cycle started Thu Dec  5 15:50:25 2002
<GC(2): freed 105000600 bytes, **80% free** (108146328/134150656), in 190 ms>
  <GC(2): mark: 172 ms, sweep: 18 ms, compact: 0 ms>
  <GC(2): refs: soft 0 (age >= 32), weak 0, final 8972, phantom 0>
<AF[2]: completed in 191 ms>

.....

<AF[64]: Allocation Failure. need 32784 bytes, 2556 ms since last AF>
<AF[64]: managing allocation failure, action=1 (29588424/131004928) (1383360/3145728)>
<GC(64): GC cycle started Thu Dec  5 15:55:16 2002
<GC(64): freed 40370448 bytes, **53% free** (71342232/134150656), in 186 ms>
  <GC(64): mark: 167 ms, sweep: 19 ms, compact: 0 ms>
```
Monitoring WebSphere Application Performance on z/OS
In Example 3-3, AF[x] indicates the xth time memory allocation failure and GC(y) indicates the yth garbage collection since the server region started. The % free number in each GC trace line indicates the amount of free memory available in the JVM after the GC.

### 3.3.2 Decreasing percentage free

If the %free number decreases over a period of time, it indicates a problem in the JVM memory heap. Ultimately, the JVM keeps trying to allocate memory and keeps failing since garbage collecting cannot recall any free memory. This occurs when all objects in the JVM have references being held that cannot be released. There are two possible causes for this problem:

- One is memory leak. This is usually an application problem, which occurs when the application mistakenly causes some objects to be referenced and never released.

- The other cause is that some transactions can be long running and there is not enough memory in the JVM for these transactions to finish and object references to be released.

This problem can be resolved by increasing the JVM heap size. In WebSphere Application Server for z/OS, set the directive JVM_HEAPSIZE=xxx in the current.env file.

### 3.3.3 RMI and frequent GC

When you run RMI (not RMI-IIOP), a GC daemon is started for you. By default, this daemon forces a GC every minute. You can modify the interval by setting the interval to a larger value, such as 1 hour:

- `-Dsun.rmi.dgc.client.gcInterval=3600000`
- `-Dsun.rmi.dgc.server.gcInterval=3600000`

### 3.3.4 Graphing GC trace output

You can plot the chart from the verbose GC trace for easier analysis. An awk script sample provided at the following URL:

http://www-1.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/TD100748

This awk script formats a trace file to a semicolon-delimited condensed file, which can be directly imported to spreadsheet tools such as Microsoft Excel or Lotus 1-2-3.
The chart in Figure 3-2 was generated by the trace shown previously. The chart indicates a JVM memory problem with the application. The problem indications include the increasing frequency of memory allocation failure and garbage collection, decreasing memory collected by GC, and last, the steadily decreasing free memory percentage in JVM. This trace was generated by an application with intended memory leak.

3.4 Work Load Manager policies

WebSphere Application Server for z/OS uses the Workload Manager (WLM) to manage the number and performance of application server regions in z/OS.

- For each server defined to WebSphere, define an application environment in the WLM panels. This provides the mechanism for WLM to manage the number of server regions (address spaces) within which WebSphere applications can run.
- The response time and throughput of WebSphere transactions are managed based on their assigned service class, associated performance objectives, and availability of system resources.

Figure 3-2  Plotting garbage collection trace
Managing the number of application server regions

Each WebSphere Application Server can have one or multiple server regions per server instance. This is based on the settings defined in the WLM application environment, see Figure 3-3.

The number of server regions created depends on the WLM determination of how the work meets its performance goals, the importance of the work compared to other work in the system, the availability of system resources necessary to satisfy these objectives. It also depends on a determination by the WLM whether starting more address spaces will help achieve the objectives.

By default, the minimum number of regions for J2EE servers is one. There is no default for the maximum. You can override the maximum and minimum number of
server regions that the WLM will start with the following three parameters in the administrative console:

- Minimum Number of Instances
- Maximum Number of Instances
- Multiple Instances Enabled

You can find these parameters in Servers → Application Servers → server name → Server Instance.

For each server, you can specify Minimum Number of Instances and Maximum Number of Instances to set boundaries on how many server regions that the WLM will start.

- Minimum Number of Instances is used to start up a basic number of server regions before the day’s work arrives. This can save time in waiting for the WLM to determine that more server regions are necessary.
- Maximum Number of Instances is useful to cap the number of address spaces started by the WLM if you determine that excessive server regions can contribute to service degradation.

Transactions received by the WebSphere server control region are passed to server regions through a set of WLM queues. The number of queues is determined by the number of service classes defined, and one server region serves only one service class at a given time. To ensure that you do not limit the parallelism of execution under full load, set Maximum Number of Instances at least as large as the number of service classes defined.

If you specify Minimum Number of Instances too low, there will be less servers available than the WLM queues. The result may be a queue bottleneck under full load conditions, since the WLM may be restricted from starting enough server regions to handle the workload. As a consequence, the system may experience queuing delays in the WLM queues resulting in transactions getting elongated response time.

**Managing the performance of WebSphere transactions**

**Server region enclave classification**

This WLM classification is used for WebSphere applications that run in the server region as part of the dispatched enclave. Each WebSphere transaction is dispatched as a WLM enclave and is managed within the server region according to the service class assigned based on the CB service classification rules.

The classification is based on the following classification criteria:

- Server name
- Server instance name
User ID assigned to the transaction
Transaction class

You can assign a default transaction class for the server or server instance for the environmental variables BBOC_HTTP_TRANSACTION_CLASS or BBOC_HTTPS_TRANSACTION_CLASS.

You can further use the virtual host name, port number, or Uniform Resource Identifier (URI) template to map the HTTP request to a transaction class with a filtering file specified in the BBOC_HTTPALL_TCLASS_FILE variable, as shown in Example 3-4.

Example 3-4  Example of mapping an HTTP request to a transaction class

<table>
<thead>
<tr>
<th>TransClassMap</th>
<th>haplex1.itso.ibm.com:*</th>
<th>/webap1/myservlet</th>
<th>WASDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>TransClassMap</td>
<td>haplex1.itso.ibm.com:7080</td>
<td>*</td>
<td>WASHI</td>
</tr>
<tr>
<td>TransClassMap</td>
<td><em>:7070</em></td>
<td>/trade/*</td>
<td>WASDF</td>
</tr>
<tr>
<td>TransClassMap</td>
<td>*</td>
<td>/eITSO/*</td>
<td>WASDF</td>
</tr>
</tbody>
</table>

Figure 3-4  WLM definitions of the server regions, CB subsystem
We recommend that you define WebSphere transaction service classes using a percentage response time objective, as illustrated in Figure 3-5. It is the technical indication to the WLM of your requirement. A response time objective is usually consistent with the business requirement of a Web application. Adjust the response time value depending on the type of application.

This option automatically generates response time distribution information that is reported through the RMF workload activity report. You will find this option useful when having to troubleshoot response time issues.

![Modify a Service Class](image)

**Server region address space classification**

In addition, we recommend that you define a report class for the server address space activity. This allows you to monitor the activity run within the server region for service tasks such as garbage collection. This WLM classification is used for tasks that run in the server region under the control of the step task and not as part of the enclave.

Classify the WebSphere Application server regions with a service goal high enough so that they can effectively compete with other workloads and be given control quickly when the WLM determines they are necessary. But use an importance and velocity lower than the enclave classification, see Figure 3-6. We
recommend that you define a reporting class in order to isolate the activity into a specific workload report.

![Subsystem definition](image)

Figure 3-6  **WLM definition of the server regions, STC subsystem**

**Control region classification**

There is a certain amount of processing in the WebSphere application control regions to receive work into the system, manage the HTTP Transport Handler, classify the work, and so on. Therefore, classify control regions also in SYSSTC or a high velocity goal.
WebSphere Application Server tools

This chapter describes and discusses the following tools available in the z/OS WebSphere Application Server environment:

- Performance Monitoring Infrastructure (PMI) Interface
- Tivoli Performance Viewer (TPV)
4.1 Performance Monitoring Infrastructure

The WebSphere Application Server collects data about the run time and applications through the Performance Monitoring Infrastructure (PMI). This data is available for use by different applications and can also be viewed via the Tivoli Performance Viewer (TPV).

For detailed information about the PMI interface see *WebSphere Application Server for z/OS: V5.1 Performance Tuning and Monitoring*, SA22-7963-02 or search the WebSphere Application Server Infocenter for PMI.

4.1.1 Enabling PMI services

In order to monitor performance data through the PMI interfaces, enable PMI services through the administrative console. If you are running in WebSphere Application Server Network Deployment mode, make sure you enable performance monitoring services in the NodeAgent.

To do this, on the admin console, select **Servers → Application Servers → server → Performance Monitoring**

PMI is designed to be a low-cost framework, therefore, it can be used in production monitoring. The following shows the average performance degradation caused by different levels of monitoring:

- With all modules set to high: approximately 2% performance degradation
- With all modules set to maximum: approximately 5% performance degradation

**Performance monitoring service settings**

You can choose to make your changes permanent by modifying the parameters under the Configuration tab or nonpersistent by modifying the parameters under the Runtime tab.

**Configuration tab**

- **Startup**: Specifies whether the application server attempts to start the specified service. If an application server is started when the performance monitoring service is disabled, you will have to restart the server in order to enable it.
- **Initial specification level**: Specifies the level of PMI tracing that you want. Set the PMI specification levels by selecting the none, standard, or custom check box:
  - **None**: Provides no data collection.
— Standard: Enables data collection for all modules except enterprise bean method level data and Java Virtual Machine Profiler Interface (JVMPI) data.

— Custom: Allows customized settings for each module. If you choose this option, you can change the level for each individual PMI module. You can set the level to none, low, medium, high, or maximum (N, L, M, H, or X), as shown in Figure 4-1.

![Performance Monitoring Settings](image)

Figure 4-1  Performance monitoring levels

**Runtime tab**

- Specifications: Specifies the PMI module and monitoring level that you have set.

Set the PMI specification levels by selecting the none, standard, or custom check box:

- None: If you choose this option, all PMI modules are set to the none level.
- Standard: Choosing this option sets all PMI modules to high and enables all PMI data excluding the method level data and JVMPI data.
– Custom: This option allows you to change the level for each individual PMI module. You can set the level to none, low, medium, high, or maximum (N, L, M, H or X). Note that you should not change the module names.

These changes will take effect immediately, but will not be persistent. Use the Configuration tab for a persistent change.

4.2 Tivoli Performance Viewer

The Tivoli Performance Viewer (TPV) is a Graphical User Interface (GUI) performance monitor that is shipped with WebSphere Application Server. It can be installed on a number of different platforms such as Windows®, AIX®, and UNIX.

The Tivoli Performance Viewer with CPU and memory utilization monitors lets you perform the following actions:

► View data in real time
► Record current data in a log and replay the log later
► View data in chart form and visually compare multiple counters. You can also scale each counter independently to enable meaningful graphs.
► View data in tabular form
► Compare data for single resources to aggregate data across a node

For detailed information about the Tivoli Performance Viewer see WebSphere Application Server z/OS V5.1 Performance Tuning and Monitoring, SA22-7963-02 or search the WebSphere Application Server Infocenter for Tivoli Performance Viewer.

4.2.1 Installing the Tivoli Performance Viewer

The Tivoli Performance Viewer (TPV) comes with your z/OS WebSphere Application Server. For this book, we installed TPV on windows machines, which involved downloading the code (80 meg) and running the installation exec on our windows machine.

4.2.2 Starting the Tivoli Performance Viewer

To start the TPV, select Start → Programs → IBM WebSphere Client Development kit for zOS → Performance Viewer.
The TPV detects which package of WebSphere Application Server you are using and connects using the default Remote Method Invocation (RMI) connector port, see Figure 4-2.

![Tivoli Performance Viewer](image)

Figure 4-2  Tivoli Performance Viewer

If the connection fails, a dialog opens to provide new connection parameters, see Figure 4-3. The Connector Type can be either SOAP or RMI.

- 8880 is the default Base port for SOAP connector
- 2809 is the default Base port for RMI connector
- 8879 is the default ND port for SOAP connector
- 9809 is the default ND port for RMI connector

![TPV connection dialog window](image)

Figure 4-3  TPV connection dialog window
4.2.3 Viewing Tivoli Performance Viewer data

The following PMI modules are available to provide statistical data:

- Enterprise JavaBeans counters
- Java Database Connectivity (JDBC™) connection pool counters
- Java 2 Connector (J2C) connection pool counters
- Servlet session counters
- Transaction counters
- Web application counters
- Dynamic cache counters
- Web services counters

**Enterprise bean module, enterprise bean, methods in a bean**

Data counters for this category report load values, response times, and life cycle activities for enterprise beans. Examples include the average number of active beans and the number of times bean data is loaded or written to the database. Information is provided for enterprise bean methods and the remote interfaces used by an enterprise bean. Examples include the number of times a method is called and the average response time for the method.

**Java Database Connectivity connection pools**

Data counters for this category contain usage information about connection pools for a database. Examples include the average size of the connection pool or number of connections, the average number of threads waiting for a connection, the average wait time in milliseconds for a connection, and the average time the connection is in use.

**Java 2 Connector connection pool**

Data counters for this category contain usage information about the Java 2 Platform, Enterprise Edition (J2EE) Connector architecture that enables enterprise beans to connect and interact with procedural back-end systems, such as Customer Information Control System (CICS), and Information Management System (IMS). Examples include the number of managed connections or physical connections and the total number of connections or connection handles.

**Servlet session manager**

Data counters for this category contain usage information for Hypertext Transfer Protocol (HTTP) sessions. Examples include the total number of accessed sessions, the average amount of time it takes for a session to perform a request, and the average number of concurrently active HTTP sessions.
Java transaction API (JTA)
Data counters for this category contain performance information for the transaction manager. Examples include the average number of active transactions, the average duration of transactions, and the average number of methods per transaction.

Web applications, servlet
Data counters for this category contain information for the selected server. Examples include the number of loaded servlets, the average response time for completed requests, and the number of requests for the servlet.

Dynamic cache
Data counters for this category contain information for the dynamic cache service. Examples include in-memory cache size, the number of invalidations, and the number of hits and misses.

Web services
Data counters for this category contain information for the Web services. Examples include the number of loaded Web services, the number of requests delivered and processed, the request response time, and the average size of requests.

4.2.4 Replaying Tivoli Performance Viewer logs
Data captured during online monitoring may be written to a log file for later analysis with TPV. This provides a very useful after-the-fact analysis tool. Use the File ? Log option from the GUI to select the log to be viewed. After the log is loaded, it can be played or fast-forwarded (time-lapse viewing). There is a rewind feature, but it does not play the data backwards, it merely moves to the beginning of the log file.

The log replay option is very useful in conjunction with the following options under Settings:

▶ Clear Buffer: You can use this option in conjunction with the Reset to Zero option. In gathering average response times, the averages must be calculated and re-averaged with historical data in buffers. This option allows the user to clear the history to take new averages in a more restricted time frame.

▶ Reset to Zero: This option removes the counts, which allows the user to select a starting point in time. Without this option, the transaction counts and average response times are accumulated from the time the monitor was started. In effect, this option allows a new starting time to be set.
- Set Buffer Size: This option is used for the graphic displays. It sets the x-axis scale for the graphs (the x-axis consists of points in time).
- View Data As: This useful feature (see Figure 4-4) allows you to view values as raw data, as changes in value, or as the rate of change in the values.

![Tivoli Performance Viewer (memory leak tpv_1207_0527.xml)](image)

**Figure 4-4  “View Data As” feature**

### 4.2.5 Tivoli Performance Viewer summary reports

TPV provides the following summary reports for each application server:

**Enterprise beans**

Enterprise beans show the total number of method calls, average response time, and multiplication of total method calls by average response time for all the enterprise beans in a table. They provide a sorting feature to help you find which enterprise bean is the slowest or fastest and which enterprise bean is called most frequently.
Enterprise JavaBeans methods
Enterprise JavaBeans (EJB) methods show the total number of method calls, average response time, and multiplication of total method calls by average response time for the individual EJB methods in a table. They provide a sorting feature to help you find which EJB method is the slowest or fastest and which EJB method is called most frequently.

Servlets
Servlets show the total number of requests, average response time, and multiplication of total requests by average response time for all the servlets in a table. They provide a sorting feature to help you find which servlet is the slowest or fastest and which servlet is called most frequently.

Web container pool
Web Container Pool shows charts of pool size, active threads, average response time, and throughput in the Web container thread pool.

Object Request Broker thread pool
Object Request Broker (ORB) thread pool shows charts of pool size, active threads, average response time, and throughput in the ORB thread pool.

Connection pool
Connection pool shows a chart of pool size and pool in use for each data source.
IBM has several tools that provide insight into applications running in a WebSphere Application Server environment. These tools include:

- Tivoli OMEGAMON for WebSphere Application Server
- Tivoli Monitoring for Transaction Performance
- WebSphere Studio Application Monitor
5.1 Tivoli OMEGAMON XE

Tivoli OMEGAMON XE for WebSphere Application Server was acquired from Candle® Corporation, and WebSphere Studio Application Monitor was acquired from Cyanea. We expect that IBM will consolidate these products into a single offering sometime in 2005. In the meantime, while there is some overlap in functionality, each tool provides unique benefits.

Tivoli OMEGAMON for WebSphere Application Server

Tivoli OMEGAMON for WebSphere Application Server is a member of the Tivoli OMEGAMON family of monitors. These monitors share a common infrastructure that allows data from many different platforms and subsystems to be shared. These monitors include:

- Tivoli OMEGAMON for z/OS
- Tivoli OMEGAMON for Customer Information Control System (CICS)
- Tivoli OMEGAMON for Information Management System (IMS)
- Tivoli OMEGAMON for DB2
- Tivoli OMEGAMON for Mainframe Networks
- Tivoli OMEGAMON for UNIX System Services
- Tivoli OMEGAMON for Storage
- Tivoli OMEGAMON for WebSphere MQ
- Tivoli OMEGAMON for WebSphere Integration Brokers

The WebSphere Application Server environment is very complex, and can be affected by many different factors. Therefore, having visibility into connected systems and subsystems is critical.

The OMEGAMON family of monitors uses a multi-tier, multi-platform architecture for scalability and flexibility. OMEGAMON consists of a data server (Candle Management Server®), one or more monitoring agents, a presentation server (Tivoli Management Portal Server), and one or more user interface clients (Tivoli Management Portal). These components can be arranged in three tiers: agent, server, client. They can also be arranged in four tiers: agent, server (hub), server (remote), client, see Figure 5-1. The second server allows for additional scalability by offloading the hub server.

The OMEGAMON family of monitors is intended to be used in a production environment, and, therefore, have capabilities suited to day-to-day operations. The dashboard provides a high-level view, showing alerts that have been generated. Alerts have associated Expert Advice to help less experienced users determine the correct course of action. Take Action commands are available to operational tasks such as starting and stopping Application Tracing or server instance.
5.1.1 Agents

OMEGAMON agents (see Figure 5-2) collect operating system, database, subsystem, and application metrics. Agents are installed on the platforms or LPARs as the resources being monitored. On z/OS, these agents are installed using System Modification Program/Extended (SMP/E). These agents pass data to the Candle Management Server when requested. When a user requests a report, the request is passed to the server, which, in turn, requests data from the agent. Ongoing data transfer between the server and agent is minimized.

Thresholds for metrics are passed from the server when the agent connects to the server. Alerts are triggered when thresholds are exceeded. Evaluation of the thresholds is performed as close to the source of data as possible. Therefore, the metrics are compared to the thresholds at the agent level, and events at the Candle Management Server are raised only when thresholds are exceeded. Automated responses to events can also be executed at the agent without communicating with the server. Thus, the agents are considered Intelligent Remote Agents.
5.1.2 Candle Management Server

The Candle Management Server collects data from the various agents. Thresholds are defined and stored at the server and distributed to the agents when the agent connects to the server. The Candle Management Server can run on Windows, AIX, HP-UX, Solaris™, and z/OS. The server runs equally well on all of these platforms. Installation on Windows tends to be quicker and easier. Implementation on z/OS has two advantages: use of SMP/E and use of Resource Access Control Facility (RACF®). Using SMP, maintenance is more reliable, since prerequisite and corequisite maintenance is checked and validated. Use of RACF security allows for validation of user access, which is then correlated with user authority. This will be a single source of user access authentication.

5.1.3 Tivoli Management Portal Server

The Tivoli Management Portal Server connects to the Candle Management Server. It analyzes and formats data for presentation at the Tivoli Management
Portal Client. The Tivoli Management Portal Server is installed on a Windows platform.

5.1.4 Tivoli Management Portal Client

The Tivoli Management Portal Client (see Figure 5-3) acts as both a display console and as an administrative console. It displays reports containing charts, graphs, and tables of data representing the health of systems and applications. It also allows a user to define new users, define thresholds to generate alerts, and customize reports, combining metrics to provide information.

The Tivoli Management Portal Client can be either a browser-based client or a Java desktop client. If you choose the browser-based client, install Java, if it is not already installed. You do not need to install other software on the client workstation. This light client model allows for easier client maintenance. As new versions of Tivoli Management Portal Client become available, the new software is automatically deployed when the browser connects to the Tivoli Management Portal Server. As an alternative, the Tivoli Management Portal Java client can be installed on the client desktop.
The Tivoli OMEGAMON Monitoring Agent® for WebSphere collects data from WebSphere’s Performance Monitoring Infrastructure (PMI), byte code instrumentation, verbose garbage collection trace, application server log, System Management Facility (SMF) 120 records, and configuration files.

## From PMI
- Application Server summary
- Application Server
- Enterprise JavaBeans (EJB) container
- Container transactions
- Container object pools
- EJ Bs
- EJB methods
From Byte Code Instrumentation

- Database connection pools
- Web applications
- Servlets/JSP Dynamic Cache (WebSphere Application Server V5)
- Workload Management (WebSphere Application Server V5)
- Servlet sessions (WebSphere Application Server V5)
- Thread pools (WebSphere Application Server V5)
- J2C connection pools (WebSphere Application Server V5)

- Workload analysis
  - EJB home delay
  - EJB method delay
  - EJB remote delay
  - Java Message Service (JMS) delay
  - Java Naming and Directory Interface™ (JNDI) delay
  - JTA delay
  - Miscellaneous delay
  - Network socket delay
  - Structured Query Language (SQL) connection delay
  - SQL query delay
  - SQL update delay
  - Servlet delay
  - User defined delay
  - EJB local home delay
  - EJB local object delay
  - CTG delay
  - Heap allocation delay

- Java Message Service
  - Queue Manager and queue names
  - Get | Put | Browse Rate
  - Average Get | Put | Browse Rate
  - JMS topic subscriber and publication message type

- Application trace
  - Server instance name
  - Start time
  - Workload type
  - Class name (workload being traced)
  - User ID and Internet Protocol (IP) address
  - Response time
  - Delay percent

- Hypertext Transfer Protocol (HTTP) sessions
- Creation date and time
- Creating user ID and IP address
- HTTP session size

► From verbose garbage collection trace:
  - Garbage collection

► From WebSphere log stream
  - WebSphere Application Server error messages
  - Who used the message (server instance name, JOBNAME, ASID, PID)
  - Message number (that is, BBOU0012W)
  - Message text

► From configuration files:
  - Counts of defined and active server instances
  - Release and build level
  - Control region processor name
  - Server region processor name
  - LDAP server information and status
  - Server type

► From SMF 120:
  - Application Server instance interval statistics
  - MOFW containers
  - MOFW classes
  - MOFW methods
  - Java 2 Platform, Enterprise Edition (J2EE) containers
  - J2EE beans
  - J2EE bean methods
  - Servant regions heap statistics

5.2 Tivoli Monitoring for Transaction Performance

Tivoli Monitoring for Transaction Performance can generate simulated transactions to measure response time, as well as measure true end user response time. This allows it to be used both in preproduction quality assurance (QA) testing to assess an application’s scalability and in production to detect response time problems. Tivoli Monitoring for Transaction Performance uses the data generated from the Open Management Group (OMG) Application Response Measurement (ARM) instrumentation. Using this instrumentation, transaction response time is broken down into its components, showing the greatest contributors to elongated response times. This includes decomposition of Java Connection Architecture (JCA) calls to CICS, IMS, and SAP. Network delays can also be identified in the transaction topology view, see Figure 5-4.
5.2.1 Application Response Measurement instrumentation

More and more applications, including middleware such as Siebel V7.7, WebSphere Application Server V5.x, and database servers such as DB2, are being ARM instrumented. Tivoli Monitoring for Transaction Performance also dynamically inserts its own ARM instrumentation into applications, eliminating the need to modify source code, and allowing for dynamic discovery of transaction topologies.

5.2.2 Agents

Agents are installed on the platforms or logical partition (LPAR), which are part of the application flow. If that platform is z/OS, SMP/E is used to perform the installation. Platforms include:

- AIX
- HP-UX
- Solaris
- OS/400®
5.2.3 Management Server

The Management Server is a shared component, and typically, only one is installed. This server collects data from the various agents that are installed. It utilizes DB2 and WebSphere Application Server. These are installed during the installation process. Platforms where the Management Server can be installed are:

- AIX
- HP-UX
- Solaris
- Red Hat Linux on xSeries, pSeries, iSeries, or zSeries
- SuSE Linux on xSeries, pSeries, iSeries, or zSeries
- Windows

5.2.4 Simulated Transaction Investigator

Tivoli Monitoring for Transaction Performance includes a stand-alone component, the Simulated Transaction Investigator (STI), which records Web transactions and plays them back. During playback, an ARM correlator is inserted to facilitate transaction correlation.

5.3 Using WebSphere Studio Application Monitor

The WebSphere Studio Application Monitor (WSAM) is part of Tivoli’s Composite Application Management solutions, which provide monitoring for WebSphere Application Server and J2EE applications. It is a problem diagnostic and performance management tool for J2EE applications, and is focused on the design and deployment phases of the application development life cycle. WSAM also helps you manage your WebSphere Application Server environment by allowing you to create, duplicate, edit, and delete Server Groups.
WebSphere Studio Application Monitor consists of a Managing Server and one or more Data Collectors. The Data Collectors run on the J2EE application server, collecting information and passing that information to the Managing Server. The Managing Server has a Kernel, which controls the environment, a Publish Server, which publishes the data, a Relational Database, and a User Interface. Figure 5-5 illustrates the monitoring levels of the WSAM.

![Figure 5-5 WSAM monitoring levels](image-url)
Managing Server

The Managing Server is designed for high scalability, and can use a load balancing approach, which distributes load across several servers. It runs on high-end UNIX or Linux servers. It consists of a database, a J2EE application server (for the user interface), and Java components (specifically Kernel, Publish Server, Archive Agent, Message Dispatcher, Polling Agent, Global Publish Server, and the Port Consolidator), as shown in Figure 5-6.

![Figure 5-6  WSAM component diagram](image)

The Managing Server can be installed on:

- AIX 5.1
- AIX 5.2
- RHEL 2.1 AS (Intel®)
- Solaris 8
- Solaris 9
- SuSe 8.1 (zSeries)

Recommended hardware is a 4-way processor with 4 GB of memory.
Data Collector
The Data Collector is a platform and application server specific code. It gathers data and forwards it the Managing Server. There are Data Collectors for WebSphere Application Server, CICS, and IMS running on z/OS. These Data Collectors are delivered as .tar files. Install them by using SMP/E, but they can also be installed without SMP/E.

Reports
Data presented by WSAM includes:
- Availability
- Enterprise overview
- Group overview
- Server overview
- Web Server overview
- Portal overview
- Server statistics overview
- Recent activity display
- System resources
- System resource comparison
- Workload Manager

Problem determination
- In-Flight request search
- Server activity display
- Memory diagnosis
- JVM thread display
- Software consistency check
- Trap and alert management

Performance Management
- Performance analysis and reporting
- Daily statistics
Figure 5-7 provides an overview of the WSAM application.

*Figure 5-7  WSAM application overview display*
Part 3

Scenarios

This section documents the scenarios that we run and their results.
Overview of the scenarios

This chapter provides an overview of the scenarios that we run.
6.1 Scenarios overview

We select six scenarios that represent problems that our customers have experienced. Given the limited amount of time available for this project and the complexity of installing and using several different tools, we create a simplistic rather than a realistic environment.

In the real world, more than one application will be running, and instead of using a base application server, we deal with a network deployment with multiple nodes. However, we believe that the underlying principles behind our scenarios can be extended to a more complex environment.

6.1.1 Tools used

We use the appropriate tools for each of the scenarios. We approach the problems initially from a system perspective. Although Resource Measurement Facility (RMF) can be very helpful in quantifying the impact of a problem, it does not allow you to drill down within an application such as OMEGAMON and other tools.
Scenario 0: Baseline

Before we start running the scenarios, we use our tools to monitor our environment and establish a baseline that we can compare to our scenarios.
7.1 Overview

It is important to measure and understand what the normal characteristics of a given workload are, before you actually have a performance problem. This allows you to make a quantitative comparison between what is a normal operation and a specific performance problem.

Without a baseline to compare to, the data provided by the monitoring tools can be just meaningless numbers. A response time of .700 seconds may be considered normal for one application, while .350 will be normal for another. If you do not determine what normal performance is for your environment, you cannot quantify how bad a performance problem is.

7.1.1 Number of baseline runs

One baseline run is not sufficient to establish what is normal behavior for any given application. We make several baseline runs using our performance monitoring tools. We save and analyze the information for each run. We run each baseline for an hour to provide us what we consider enough data to analyze.

During the course of these baseline runs, we find several subtle variances in performance and track them back to normal occurrences, such as garbage collection (GC). Many factors can come into play, such as a majority of users establishing sessions at the beginning of a work day or after a lunch break.

Because the WebSphere Application Server for z/OS competes for resources on the z/OS system, you also need to understand the impact that other applications can have. While the Workload Manager (WLM) attempts to divert resources to the highest priority tasks, the performance of a given transaction may still vary, depending upon the time of day because of other priority loads placed upon the system.

Several hardware changes are also made during the course of our testing and we find that our performance has improved during the last few scenario runs. At this point, we rerun our baseline and adjusted our comparison of the later scenarios to match the current environment.

Important: Even after establishing a baseline, you must periodically go back and re-validate or re-establish the baseline.

7.1.2 Controlled environment

For the purpose of this book, we use WebSphere Workload Simulator (WSWS) and a set of scripts to run a controlled environment with 100 simultaneous users
for our baseline scenario. Although it is important to measure and establish the baseline for a real production environment, it is also worthwhile to establish a baseline for a controlled environment that can be used for a later comparison.

In our example we do not have a real production environment, so all we can measure is a controlled environment. However, the principle is the same. We also compare the controlled environment baseline to a controlled environment problem. In the real world, you have to compare a production baseline to a production problem.

Finally, it is also important to understand that no one tool can provide all the answers. We use a combination of tools to establish our baseline and decide what is normal operation for our application.

### 7.1.3 Baseline characteristics

In executing several runs on two different set of servers, the baseline performance characteristics we document are listed in Table 7-1.

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Low</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction ended</td>
<td>15000</td>
<td>16000</td>
</tr>
<tr>
<td>Transaction END/S</td>
<td>50</td>
<td>55</td>
</tr>
<tr>
<td>Actual time</td>
<td>.150</td>
<td>.220</td>
</tr>
<tr>
<td>Execution time</td>
<td>.045</td>
<td>.075</td>
</tr>
<tr>
<td>Queued time</td>
<td>.080</td>
<td>.150</td>
</tr>
</tbody>
</table>

- Application CPU utilization: Between 30% and 45%
- No regular system paging
- Little or no resource delays
- Total system CPU utilization less than 90%
- Garbage collection intervals of 90 seconds to 120 seconds and completed in 590 ms to 830 ms.
7.2 RMF displays for baseline scenario

When you first notice a performance problem, use the Resource Measurement Facility (RMF) Monitor III displays to interactively view some of the system's performance data. We set the interval for the interactive RMF Monitor III display at 100 seconds so that it will give us a fairly accurate idea of what happens in the last minute or so.

We use these displays to check our performance index (PI) trigger and also to quantify the impact of the performance problem. Some of the information is repeated on different screen displays. It is not necessary to use all of the displays to understand the impact of the problem, but we provide all of them so that you can see the differences.

The primary screens used for the baseline and subsequent scenarios are:

- Sysplex summary (SYSSUM)
- Response time distribution (SYSSRTD)
- Work Manager delays (SYSWMD)
- System information (SYSINFO)

In addition, we also use the following screens:

- Add screen name
- Add screen name
7.2.1 Sysplex summary

We initiate several sysplex summary displays during the course of our baseline runs. Figure 7-1 represents one of these sysplex summary displays.

For our baseline run, our goal is to have 95% of our transactions complete in less than .700 seconds. Figure 7-1 shows that 96% of our transactions meet this criteria.

![RMF sysplex summary for baseline scenario](image)

**Figure 7-1**  RMF sysplex summary for baseline scenario

**Performance index**

The performance index or Perf Indx should equal 1.0. Anything higher indicates that you are not meeting your goals. If the performance index is less, it means you are exceeding your goals. In this case we are at .90, which means that we are slightly exceeding our goal.

The performance index serves as a trigger to indicate that something is wrong. It needs to be sensitive enough to indicate when things are starting to go wrong, but not so sensitive that it constantly alerts you to changing conditions.

We provide several displays that occurred during the course of each of our baseline runs and average our results, which are listed in Table 7-2. We consider these results as normal for our application.

**Table 7-2  Baseline performance index**

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>95%</td>
<td>96%</td>
<td></td>
</tr>
<tr>
<td>Performance index</td>
<td>1.00</td>
<td>.90</td>
<td></td>
</tr>
</tbody>
</table>
Throughput
Table 7-3 shows that our transaction throughput, called *Trans Ended Rate* is 54.97. Our throughput varied between 50 and 60 transactions per second when running 100 users.

*Table 7-3  Baseline throughput*

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction ended rate</td>
<td>50</td>
<td>60</td>
<td>54.97</td>
</tr>
</tbody>
</table>

Response time
The total time, called *Actual*, is made up of the WAIT and EXECUT time. In this display (see Table 7-4), our application is still ramping up to 100 users, so our performance is slightly better than normal at this point.

*Table 7-4  Baseline response time*

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait time</td>
<td>.080</td>
<td>.100</td>
<td>.50</td>
</tr>
<tr>
<td>Execution time</td>
<td>.045</td>
<td>.075</td>
<td>.39</td>
</tr>
<tr>
<td>Actual time</td>
<td>.125</td>
<td>.175</td>
<td>.90</td>
</tr>
</tbody>
</table>

Throughout this book, we provide tables to compare our scenarios to the baseline runs.

Note that we use the RMF displays to just check the high level symptoms. These displays show what is happening with the Application Server, but do not allow you to drill down within an application.

Since the performance monitoring tools such as OMEGAMON, WebSphere Studio Application Monitor (WSAM), or Tivoli Performance Viewer (TPV) may not be active when a problem first occurs, these displays may be the only source of immediate information. Therefore, you should understand what the RMF displays and reports can tell you about any problem you are experiencing. If the problem persists, the next logical step may be to bring up one of the monitoring tools and check the individual applications involved.

In addition to using the RMF displays, it is important to check the RMF reports for the time period involved. These reports can provide more details about the Application Server environment and can offer other clues to the root cause of a performance problem.
7.2.2 Response time distribution

Figure 7-2 displays the same information for the *wait, execution, and actual* times as shown in Figure 7-1 on page 75. It also shows the same information as the Transaction Ended Rate in the sysplex summary display, but in this case, it is called *Trx Rate*.

In addition, Figure 7-2 also provides a graphical representation of the transaction response times. We use this display to both quantify the problem and to understand how the poor performance response time is distributed.

![Figure 7-2 RMF response time for baseline scenario](image)

**Response time and throughput**

You can also find this type of information in an RMF report (see 7.3.1, “Workload activity report” on page 83) and in a WSAM display (see 7.6.5, “WLM service class period detail” on page 104).

There is a lot of overlap among the tools that we use, therefore we recommend that you use the tools and displays that you are most comfortable with. For example, people with a z/OS background may prefer the RMF display, while others with a distributed background may prefer to use the WSAM display.
Even within a given tool, there are a number of ways to display the same information, just as both the sysplex summary and response time distribution displays contain similar statistics. Again, we provide a table of what we consider the normal values for this type of display. The values are the same as the previous table, only the name of the throughput field is changed to match the display.

### 7.2.3 Work Manager delays

You can also see almost the same information in the Work Manager delays display, see Figure 7-3.

![RMF Work Manager delays for baseline scenario](image)

**Figure 7-3**  RMF Work Manager delays for baseline scenario

#### Throughput

Compute the throughput or transaction rate per second of 54.97 by taking the number of transactions (for 5497 TRX) and dividing it by 100 seconds, which is the time period for this report.

#### Response time

The average response time shown as Avg. Resp. time in Figure 7-3 is equivalent to the actual time shown in other displays. The execution time is shown as the average Exec time in Figure 7-3 and the wait time is not shown.

Apart from allowing you to quantify the impact of a problem, this display also allows you to see if there are any delays that can potentially cause a performance problem.
Using Table 7-5 we compare what we consider the normal output for this display with each of our scenarios.

**Table 7-5 Baseline Work Manager delays**

<table>
<thead>
<tr>
<th></th>
<th>Low</th>
<th>High - normal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average response time</td>
<td>.125</td>
<td>.175</td>
</tr>
<tr>
<td>For ---- transactions</td>
<td>5000</td>
<td>6000</td>
</tr>
<tr>
<td>Average execution time</td>
<td>.045</td>
<td>.075</td>
</tr>
</tbody>
</table>

7.2.4 System information

The system information display (see Figure 7-4) shows a combination of response time, throughput, CPU utilization, and delays for the service class on a specific logical partition (LPAR).

![Figure 7-4 RMF system information for baseline scenario](image)

**Throughput**

For the system information display the *actual time* is shown as RESP Time and throughput is shown as TRANS/SEC. Though the names may change between the different displays, the data is the same.

Use this display to quantify the impact of a performance problem and also see if there are any delays that can potentially cause a problem.
### Response time
This display comes later in the baseline run, and shows an average response time of 2.64 seconds and a throughput of 21.94 tps. This is not indicative of our baseline run for this application. We traced this dip in performance to the fact that garbage collection was running during this display.

**Note:** It is important to understand that peaks and valleys occur in every workload.

When we check the system information displays taken during other baseline runs, we find that the output listed in Table 7-6 is representative of what is normal.

| Table 7-6  Baseline system information |
|-----------------|-----------------|
| **Baseline**    | **Low**         | **High - normal** |
| Response time   | .125            | .175              |
| Transaction/SEC | 50              | 60                |

This display serves to illustrate the fact that not only must your performance monitoring be quantitative, it must also be consistent over a period of time. A single display of a 100-second time slice may not give you representative data.

### 7.3 RMF reports for baseline scenario
The RMF displays are helpful to see what is occurring at any given moment, that is, to monitor performance. You also have to measure and analyze them over a longer period of time. One of the best ways to do this and determine the baseline information for an application is to check RMF reports, in particular, the workload activity report, which can be a valuable source of information.

One important difference between this report and the preceding displays is that each entry in the report covers a 5-minute period and there are multiple entries, which span the entire baseline run. The reports represent consistent historical data, though the displays are transient in nature and lack consistency because of the random frequency of execution.
7.3.1 Workload activity report

In the previous section, we set a response time goal that 95% of our transactions must be complete in .700 seconds. Figure 7-5 shows the workload activity report for the response time distribution.

![Response time distribution table]

**Performance index**

In this case, you can see that we have an ACTUAL% of 95.0%. Therefore, we are right on target and that is reflected in our performance index or PERF INDX of 1.

**Response time distribution**

The response time distribution see Figure 7-5) shows a graphical breakdown of the transaction response times for the preceding 5 minutes. It shows that 91.4 of the transactions took less than .350 seconds.

As noted previously, this type of information can be provided by both an RMF display (see 7.2.2, “Response time distribution” on page 79) and a WSAM display (see 7.6.5, “WLM service class period detail” on page 104).
7.3.2 Workload activity transactions

The workload activity report for the WASPM service class (see Figure 7-6) provides more detailed information than the RMF displays and some of the additional information can prove valuable in determining the cause of a performance problem.

Figure 7-6   RMF workload activity report: Transactions for baseline scenario

Figure 7-6 shows the same type of information that the RMF system summary displays.

Throughput

Figure 7-6 shows that 15750 transactions ended during this 5-minute time period. The throughput or END/S is 52.43. These numbers are fairly typical for this application. Table 7-7 lists the throughput.

Table 7-7   Workload activity report throughput

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction ended</td>
<td>15000</td>
<td>16000</td>
<td></td>
</tr>
<tr>
<td>END/S</td>
<td>50</td>
<td>60</td>
<td></td>
</tr>
</tbody>
</table>

Response time

The actual transaction time of 1.38, which consists of both the execution time of .049 and the wait time, shown as QUEUED time, of .088 are also typical for our application.
The transaction times are in the same range as those reported in our RMF display, see 7.2.1, “Sysplex summary” on page 77.

Table 7-8  Workload activity report response time

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual time</td>
<td>.125</td>
<td>.175</td>
</tr>
<tr>
<td>Execution time</td>
<td>.045</td>
<td>.075</td>
</tr>
<tr>
<td>Queued time</td>
<td>.080</td>
<td>.100</td>
</tr>
</tbody>
</table>

DASD input/output

Also note that our application servant region does little or no DASD input/output (I/O), see Table 7-9. This is normal for a lot of Web applications, where the I/O is usually done on behalf of the enterprise information system (EIS) back end, such as DB2, Customer Information Control System (CICS) or Information Management System (IMS).

Table 7-9  DASD I/O times

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSCHRT</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Resp</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Connect</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Disc</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

Also note that there is no storage or paging information shown for the application servant region. This is also normal, the storage information and paging information, if any, is typically shown in a section of the report related to the controller region.
7.3.3 Workload activity WASPC

Although we do not show the controller region information for each of scenarios in this book, we check that information to determine what we consider is normal for when our application is running.

Figure 7-7 shows a high paging rate, which is caused by garbage collection running with a large Java virtual machine (JVM) heap size.

![RMF workload activity report: WASPMC for baseline scenario](image-url)
7.3.4 System summary

We also use the system summary report (see Figure 7-8) to track how busy the central processing unit (CPU) is and whether or not we are paging excessively.

![RMF System Summary Report](image)

Figure 7-8 RMF system summary report for baseline scenario

As you can see there is some variation during the course of our baseline run. We averaged around 35% to 45% for CPU busy and our demand paging rate is normally under 1.0. Table 7-10 lists the CPU busy and demand paging output.

Table 7-10 System summary report: CPU busy and demand paging

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU busy</td>
<td>35%</td>
<td>45%</td>
</tr>
<tr>
<td>Demand paging</td>
<td>.30</td>
<td>3.0</td>
</tr>
</tbody>
</table>
7.4 Tivoli Performance Viewer for baseline scenario

The Tivoli Performance Viewer (TPV) monitor is run with logging turned on while collecting the baseline data. After the run, we use the log replay function to analyze the baseline. For more information, see 4.2.4, “Replaying Tivoli Performance Viewer logs” on page 51. TPV provides metrics such as response times, transaction volumes, and pool sizes.

7.4.1 Servlets

Figure 7-9 shows that approximately 1500 servlets are being processed every 10 seconds. The response time fluctuates, generally staying between 9 ms and 20 ms. The differences in response times are influenced by the intervals in which garbage collection occurs.

![Figure 7-9 TPV servlet statistics for baseline scenario](image-url)
### 7.4.2 JVM runtime

In Figure 7-10 the JVM memory shows used memory between 20 KB and 40 KB, with garbage collection roughly once a minute.

![JVM runtime memory for baseline scenario](image)

Figure 7-10  JVM runtime memory for baseline scenario
7.4.3 Object Request Broker pool summary

The Object Request Broker (ORB) summary (see Figure 7-11) shows the number of active threads fluctuating throughout the period, but usually has a value less than 4. The average method response time slowly dropped from 120 ms to roughly 80 ms, with a throughput rate of slightly under 40/second.

![Figure 7-11 ORB pool display for baseline scenario](image)
7.5 OMEGAMON for baseline scenario

For the scenarios, we use two monitors of the OMEGAMON suite of products, the Multiple Virtual Storage (MVS) operating system (OS) monitor for system level monitoring, and the WebSphere Application Server OS/390® monitor for server monitoring. These products are highly customizable in the building of displays. However, we choose to stay with predefined displays with very little modification.

When you are familiar with these products, develop customized displays to reduce the number needed to get the desired information. The primary displays used in the baseline and the scenarios include:

► For the MVS operation system monitor:
  – WLM service class resources
  – Address Space CPU utilization
  – System paging activity
  – Address space bottlenecks

► For the WebSphere Application Server OS/390:
  – Servant regions
  – All workloads
  – Web applications
  – EJB containers
  – Garbage collection activity
  – Heap usage
  – Suspected memory leaks

OMEGAMON also has a great deal of customization for setting triggers on various screens and a notification process to alert the user when a triggered event has occurred. For the purpose of the scenarios, we set only one trigger, which is for the PI at two levels: One when the PI is greater than 1.5 and another when the PI is greater than 4.
7.5.1 WLM service class resources

Figure 7-12 shows where the PI trigger is set and shows the PI, the average response time, CPU percentage and other statistics related to a WLM service class.

The service class for our server regions, WASPM, shows in the CPU Percentage graphical display as using approximately 30% CPU. The service class is not shown in the table at the bottom of the screen because the PI of the WASPM service class is below 1 for the baseline run.

![Figure 7-12 OMEGAMON: WLM service class resources for baseline scenario](image-url)
7.5.2 Servant regions

The servant regions screen (see Figure 7-13) shows the number of servant regions running, the JVM memory utilization for each region, and the number of Hypertext Transfer Protocol (HTTP) sessions for each region. In this case, we have two servant regions, both with a JVM heap size of 269154816 (256 M), and one has 47 HTTP sessions while the other has 98 sessions.

![Figure 7-13 OMEGAMON: Servant regions for baseline scenario](image)
7.5.3 Garbage collection activity

The garbage collection activity screen (see Figure 7-14) for our baseline run shows the collection rate, heap usage, and real time percentage for both of our servant regions, identified by their Address Space IDentifier or ASID.

Figure 7-14  OMEGAMON: Garbage collection activity for baseline scenario
7.5.4 Garbage collection history

Selecting a specific row in the garbage collector activity table drills down to show the garbage collection history for a specific ASID, as shown in Figure 7-15. In this case, the screen shows a very stable heap usage, a garbage collection rate of less than .5 collections per minute, and a real time percentage of less than 1%.

Figure 7-15  OMEGAMON: Garbage collection history for baseline scenario
7.5.5 Web applications

The Web Application series of screens (see Figure 7-16) shows all of the Web applications defined to the server and the average response time, total requests, and request rate for each application. For our baseline, we processed 4543 requests in the last 61-second interval with an average response time of 10.676 ms and a request rate of 74.475 tps.

![Figure 7-16 OMEGAMON: Web applications 1 for baseline scenario](image)

By selecting the desired row from the Web applications table, a drill down screen shows the same information for all servlets and JavaServer Pages (JSP™) that are contained within the selected application.
7.5.6 Web applications: Ten worst average response times

![Graph showing web application response times and request rates.](image)

Figure 7-17  OMEGAMON: Web applications 2 for baseline scenario
7.5.7 EJB containers

Another series of screens that shows detailed application information is the EJB containers screens, as shown in Figure 7-18. The first of these shows the average response time for the entire container.

By selecting the row on the EJB container table, a drill down screen shows all of the EJBs in the container.
7.5.8 EJB method invocations

![Image of OMEGAMON: EJB containers 2 for baseline scenario]

Figure 7-19  OMEGAMON: EJB containers 2 for baseline scenario
Finally, by selecting the row for a specific bean, another drill down screen shows all of the methods contained within the bean along with the number of times and the EJB: Ten worst average response times, see Figure 7-20.

Figure 7-20  OMEGAMON: EJB containers 3 for baseline scenario

7.6 WSAM for baseline scenario

The primary WSAM screens used in the baseline and later scenarios are:

- Enterprise, group, and server overviews
- Server statistics overview
- WLM service class summary and period detail
- Recent activity
- Web application summary
- EJB summary

We set trigger levels for the average response time on the server statistics overview screen.
7.6.1 Enterprise overview

The enterprise overview (see Figure 7-21) displays the availability for all the applications running on the assigned server groups. The yellow button indicates that the threshold meets or exceeds the first alert level and a red button that indicates the threshold meets or exceeds the second alert level.

![Enterprise overview](Figure 7-21 WSAM: Enterprise overview for baseline scenario)
7.6.2 Server overview

The server overview (see Figure 7-22) displays comprehensive statistics, activity, and resource data for the selected server. Use the server selection drop-down menus to select a different server.

Figure 7-22  WSAM: Server overview for baseline scenario

7.6.3 Servant regions overview

The servant regions overview series of screens (see Figure 7-23) shows the status, total volume, throughput, and response time statistics for servers at an enterprise, group, or server level. This screen graphically shows any trends or spikes in throughput or response time over a one-hour interval.

Figure 7-23  WSAM: Servant regions overview in the baseline scenario
7.6.4 Server statistics overview

Navigating from the servant regions overview to the server statistics overview (see Figure 7-24), we see in table form, many of the key metrics for a specific server plus any alerts for values exceeding a preset trigger level. In this instance, we see an average response time between 26 ms and 30 ms.

![Server statistics overview](image)

*Figure 7-24  WSAM: Server statistics overview for baseline scenario*
7.6.5 WLM service class period detail

The WLM service class period detail screen (see Figure 7-25) screens the baseline response time distribution, the total number of server transactions completed, and the distribution of response time relative to the WLM goal of 95% in 700 ms.

![Figure 7-25 WSAM: WLM service class period detail for baseline scenario](image_url)
7.6.6 Number of requests versus average response time

We use the recent activity screen for several purposes by setting different metrics to be compared over a specified period of time, as shown in Figure 7-26. First we compare the number of requests to average response time.

Figure 7-26  WSAM: Recent activity screen showing the number of requests versus average response time
7.6.7 Number of requests versus total garbage collection time

To assess garbage collection activity, we next compare the number of requests to total garbage collection time, see Figure 7-27. This shows reasonably stable garbage collection activity, with no significant impact on throughput.

Figure 7-27  WSAM: Recent activity screen showing the number of requests versus average GC time
Scenario 1: Write to hierarchical file system file
8.1 Overview

In this scenario, we modify the Trader application by adding a "write to an hierarchical file system (HFS) file" to one of the servlets. We expect this change to increase the transaction response times and demonstrate the overhead of writing data to a log file in the HFS.

This scenario represents a case where an application programmer added a "write to an HFS file" within a transaction to help debug a problem. Unfortunately, when the problem was fixed, the write was not removed and the updated code was moved onto the production system.

Our implementation of this scenario is fairly crude: Because the write was added to a common path, it meant that the impact was fairly severe, since every transaction caused a write to an HFS. In the original problem, the modified transaction was executed with much less relative frequency and was harder to identify. However, the underlying methodology of finding the problem is still valid.

8.2 RMF displays for scenario 1

When we start a scenario, we use the Resource Measurement Facility (RMF) Monitor III and display the sysplex summary screen. This highlights any service class that does not meet its performance goal.

8.2.1 Sysplex summary

The sysplex summary shows that our goal is to have 95% of the transactions completed in less than .700 ms, see Figure 8-1.

![Figure 8-1 RMF sysplex summary for scenario 1](image-url)
Performance index
The initial signal or trigger that indicates a problem is that our performance index increased to 4.00, see Table 8-1. This is caused by the fact that only 60% of the transactions meet our response time goal of .700 ms per transaction.

Table 8-1  Write to HFS performance index

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>Normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual response percentage</td>
<td>95%</td>
<td>60%</td>
<td></td>
</tr>
<tr>
<td>Performance index</td>
<td>1.00</td>
<td>4.00</td>
<td></td>
</tr>
</tbody>
</table>

We repeated this display several times, however, the performance degradation did not vary significantly.

Throughput
The next step is to check the throughput and compare it to our baseline runs. The transaction rate has dropped from approximately 50 - 60 per second to approximately 43 per second, see Table 8-2. This represents an approximate reduction of 25% in throughput.

Table 8-2  Write to HFS throughput

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction ended rate</td>
<td>50</td>
<td>60</td>
<td>42.94</td>
<td></td>
</tr>
</tbody>
</table>

Response time
We check the response times and compare them to our baseline runs, see Table 8-3. We find that the response time has increased from approximately .125 ms - .175 ms to almost .800 ms. This means that a transaction will take four times longer than the normal time to complete and represents an increase of approximately 400%.

Table 8-3  Write to HFS response time

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait time</td>
<td>.080</td>
<td>.100</td>
<td>.538</td>
<td></td>
</tr>
<tr>
<td>Execution time</td>
<td>.045</td>
<td>.075</td>
<td>.255</td>
<td></td>
</tr>
<tr>
<td>Actual time</td>
<td>.125</td>
<td>.175</td>
<td>.793</td>
<td></td>
</tr>
</tbody>
</table>
**Conclusions**

At this point, we can quantify the impact of the performance problem at the application server level. We know that the throughput has dropped and the average response time has increased. We can also see that both the execution and the wait time have increased, with the increase in wait time being slightly more than the increase in execution time.

In this scenario, note that the difference in response time is not apparent to the average user. It is difficult to tell the difference between two-tenths of a second and eight-tenths of a second. However, the increase in the performance index indicates a problem.

**8.2.2 Response time distribution**

We check the response time distribution screen (see Figure 8-2). This screen shows the response time and throughput information and also allows you to graphically see how the transactions are distributed according to response time.

![Figure 8-2 RMF response time distribution for scenario 1](image-url)
Response time and throughput
The response time and throughput in this display (see Table 8-4) are the same as for the sysplex summary display. This is to be expected since the displays are done within seconds of each other and our sample has not changed.

Table 8-4 Write to HFS response time and throughput

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction rate</td>
<td>50</td>
<td>60</td>
<td>42.94</td>
<td></td>
</tr>
<tr>
<td>Wait time</td>
<td>.080</td>
<td>.100</td>
<td>.538</td>
<td></td>
</tr>
<tr>
<td>Execution time</td>
<td>.045</td>
<td>.075</td>
<td>.255</td>
<td></td>
</tr>
<tr>
<td>Actual time</td>
<td>.125</td>
<td>.175</td>
<td>.793</td>
<td></td>
</tr>
</tbody>
</table>

Response time distribution
When we check the response time distribution, we see a shift to the right. Although some of our transactions are still meeting their performance goal, our overall distribution has spread out and a substantial number of the transactions are running significantly slower.

8.2.3 Work Manager delays
We check the Work Manager delays screen (see Figure 8-3). This screen in addition to the response time and throughput information, allows you to see whether there are any system delays.

Figure 8-3 RMF Work Manager delays for scenario 1
Throughput and response time
This display shows the same time period as the two preceding displays (see Figure 8-1 and Figure 8-2) and contains the same information for throughput and response time.

Note that the transaction rate (TRX) shown in this display is 4294. This is the number of transactions that were processed in the last 100 seconds, which equals a rate of 42.94 per second. Table 8-5 lists the scenario 1 Work Manager delays throughput and response time.

Table 8-5  Write to HFS: Work Manager delays throughput and response time

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>For ---- transactions</td>
<td>5000</td>
<td>6000</td>
<td>4294</td>
</tr>
<tr>
<td>Average response time</td>
<td>.125</td>
<td>.200</td>
<td>.793</td>
</tr>
</tbody>
</table>

Also note that the wait information is not displayed on this screen. You can determine this by subtracting the execution time from the response time.

Delayed by information
The main reason to see this screen is that it provides the Delayed by information. However, in this case, there is no indication of the cause of our problem.

8.2.4 System information
Next we check the overall system statistics (see Figure 8-4) to see if there are any z/OS system problems that can account for the response time increase.
Throughput
This screen, taken during a different run executing the scenario, shows a similar decrease in transaction throughput. In this case, our transaction rate is 39.92 (as shown in Table 8-6), whereas in the previous displays the rate is 42.94.

Table 8-6 Write to HFS: System information throughput

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transactions/SEC</td>
<td>50</td>
<td>60</td>
<td>39.92</td>
</tr>
</tbody>
</table>

Response time
We can also see a similar increase in the response time from .793 to .987 (see Table 8-7).

Table 8-7 Write to HFS: System information response time

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response time</td>
<td>.125</td>
<td>.200</td>
<td>.987</td>
</tr>
</tbody>
</table>

Average Multiple Virtual Storage (MVS) utilization percentage
Although the CPU utilization is a little high, it is not abnormal and is not the cause of this problem.
### Conclusions

After seeing the RMF displays all we conclude that we have a performance problem, relative to our baseline. Because we controlled the changes to the system, we know the problem is caused by a new version of our application. Therefore, we temporarily back the application off of our system and check our RMF reports for the time period that the application is running.

---

### 8.3 RMF reports for scenario 1

In addition to seeing the RMF displays, it is important to check the RMF report. Check each of the five-minute intervals during the time period that we run each scenario.

#### 8.3.1 Workload activity report

The workload activity report shows our goal, which is to have 95% of the transactions completed in less than .700 ms. Figure 8-5 shows the performance goal for scenario 1.

---

**Figure 8-5  RMF workload activity report: Performance goal for scenario 1**
Performance index
In this case, only 60.4% of our actual transactions achieved this rate (see Table 8-9). Because of our application's poor performance relative to our goal, the performance index show 4.0. As with the RMF sysplex summary display, this is a trigger that indicates a problem.

<table>
<thead>
<tr>
<th>Table 8-9 Write to HFS: Workload activity performance index</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baseline</strong></td>
</tr>
<tr>
<td>Actual response percentage</td>
</tr>
<tr>
<td>Performance index</td>
</tr>
</tbody>
</table>

Response time distribution
When we check the response time distribution, we see that the distribution of the other 40% of our transactions that took more than .700 seconds.

8.3.2 Workload activity transactions
Similar to what we did previously, we further check each of the intervals at which the performance index is set to 4.0. Figure 8-6 is a representative example of one of these intervals.

| Figure 8-6 RMF workload activity report: Transactions for scenario 1 |
**Throughput**

The total number of transactions is slightly lower than the baseline of approximately 15000 - 16000 transactions. The throughput is down to almost 15% from the baseline of 50 - 60 tps. Table 8-10 shows these details.

*Table 8-10  Write to HFS: Workload activity throughput*

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transactions ended</td>
<td>15000</td>
<td>16000</td>
<td>12739</td>
</tr>
<tr>
<td>END/S</td>
<td>50</td>
<td>60</td>
<td>42.47</td>
</tr>
</tbody>
</table>

**Response time**

The queue time of 505 ms, with an execution time of 259 ms equals a total actual time of 764 ms, see Table 8-11. This is significantly higher than our baseline, but note that this is a symptom of our problem and not the problem itself.

*Table 8-11  Write to HFS: Workload activity response time*

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual time</td>
<td>.125</td>
<td>.200</td>
<td>.764</td>
</tr>
<tr>
<td>Execute time</td>
<td>.045</td>
<td>.080</td>
<td>.259</td>
</tr>
<tr>
<td>Queued time</td>
<td>.080</td>
<td>.120</td>
<td>.505</td>
</tr>
</tbody>
</table>

**DASD input/output (I/O)**

Table 8-12 shows that the DASD I/O times stand out. Normally this application does not perform any I/O.

*Table 8-12  DASD I/O times*

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSCHRT</td>
<td>0</td>
<td></td>
<td>4.7</td>
</tr>
<tr>
<td>Resp</td>
<td>0</td>
<td></td>
<td>1.4</td>
</tr>
<tr>
<td>Connect</td>
<td>0</td>
<td></td>
<td>.9</td>
</tr>
<tr>
<td>Disc</td>
<td>0</td>
<td></td>
<td>.2</td>
</tr>
</tbody>
</table>

The fact that we are performing I/O may be the reason for the symptoms in this scenario. The delays involved in performing I/O to the HFS add to the transaction times.
We have to investigate this further since it is potentially the cause of the problem and not a symptom. That is, transactions are running slower because we are performing I/O. It does not mean the opposite, which is that we are performing I/O because the transactions are running slower.

### 8.3.3 System summary report

The summary report for the same time period shows CPU utilization of 42.8% and minimal system paging, as shown in Figure 8-7.

![RMF SUMMARY REPORT](image)

**Figure 8-7**  System summary report for scenario 1

Table 8-13 shows the CPU busy and demand paging output.

**Table 8-13  System summary report: CPU busy and demand paging**

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU busy</td>
<td>35%</td>
<td>45%</td>
<td>39.9 - 44.3%</td>
<td></td>
</tr>
<tr>
<td>Demand paging</td>
<td>.30</td>
<td>3.0</td>
<td>0.93 - 2.78</td>
<td></td>
</tr>
</tbody>
</table>

### 8.4 TPV for scenario 1

We did not utilize the Tivoli Performance Viewer (TPV) in this scenario.
8.5 OMEGAMON for scenario 1

We use OMEGAMON to dig deeper into the application running on the WebSphere Application Server for z/OS.

8.5.1 Workload Manager (WLM) service class resources

Similar to both the RMF display and RMF reports, the performance index provides the first indication of a problem. The workload display (see Figure 8-8) shows the WASPM service class has a performance index of 4.10 and an average response time of 1.185 seconds.

![Figure 8-8 OMEGAMON WLM service class resources for scenario 1](image-url)
8.5.2 Servant regions

The servant region display (see Figure 8-9) shows the two servant regions and a total of approximately 100 Hypertext Transfer Protocol (HTTP) sessions, indicating that the incoming volume is the same as in the baseline run.

![Figure 8-9 OMEGAMON servant regions for scenario 1](image-url)
8.5.3 Garbage collection activity

The garbage collection activity display (Figure 8-10) shows a garbage collection rate below one per minute, a real time percentage of less than 1%, and plenty of available heap. All of this indicates that the change in response time is not related to memory utilization.

![Garbage Collection Activity Diagram](image)

Figure 8-10  OMEGAMON garbage collection activity for scenario 1
8.5.4 Web applications

Finally, we check the changes in the application components with the Web application and EJB container displays, see Figure 8-11.

![Figure 8-11 OMEGAMON web applications for scenario 1](image)
8.5.5 Web application drill down

After selecting our Trader application from the first Web application display, a drill-down display shows the servlets and Java ServerPages (JSP) components that make up the application. Figure 8-12 shows the Web application display identifies the components with the highest response times and we see one servlet with a 280 ms response time. Because the response time of this servlet alone is five times higher than the baseline execution time for the entire transaction, this component is probably the main cause for the change in the performance characteristics.

![Figure 8-12 OMEGAMON Web application drill down for scenario 1](image-url)
8.5.6 EJB container

Finally, the EJB container display shows, at a method level, the response times of the EJB component of the application, see Figure 8-13. Response times are shown as 15 ms or less for the EJB methods. This is consistent with the baseline and again highlights the servlet component as probably the main cause for the overall increase in transaction response time.

Figure 8-13 OMEGAMON EJB container for scenario 1
8.6  WSAM for scenario 1

In the WebSphere Studio Application Monitor (WSAM) monitor, the first indication of a performance change is in the server statistics overview screen where we have set a trigger level on average response time.

8.6.1 Server statistics overview

The server statistics overview screen (see Figure 8-14) shows both the servant regions exceeding response time limit with response times of 244 ms and 234 ms.

![Figure 8-14  WSAM server statistics overview for scenario 1](image)

8.6.2 Response time distribution

When we checking the response time distribution (see Figure 8-15) of the WASPM service class, it shows the change of distribution, with a significant increase in the number of transactions that are 200% - 400% above the goal.

![Figure 8-15  WSAM response time distribution for scenario 1](image)
8.6.3 Servlet summary

When we check the application components, first the Web application components are shown in the servlet summary, see Figure 8-16. The TraderSQLJDBCServlet shows an average response time of 324.39 ms, which is well above the baseline.
8.6.4 EJB method summary

Finally, when we check the EJB components with the EJB method summary, as shown in Figure 8-17, the average response times of the EJB methods are 35 ms or less. Again, the servlet component is the most likely cause of the response time increase.

![EJB method summary screenshot]

*Figure 8-17  WSAM EJB method summary for scenario 1*
Scenario 2: Excessive objects
9.1 Overview

In this scenario, we take the basic Trader application and modify a couple of the methods to generate a large number of objects each time they are executed. This is a fairly common programming problem that has several impacts. First, the generation of the objects makes the transaction much more central processing unit (CPU) intensive, which extends the response time. Secondly, it causes garbage collection to occur more frequently, again causing more delay and extending response times for some transactions.

9.2 RMF displays for scenario 2

When we start a scenario, we use the Resource Measurement Facility (RMF) Monitor III and display the sysplex summary screen. This highlights any service class that does not meet its performance goal.

9.2.1 Sysplex summary

The sysplex summary shows that our goal is to have 95% of the transactions completed in less than .700 ms, see Figure 9-1.

![Figure 9-1  RMF sysplex summary for scenario 2](image)

Performance index

As you can see from Figure 9-1, the impact of this problem is very severe. The performance index of **** indicates an index greater than 4, see Table 9-1. The actual response time percentage shows that only 18% of the transactions meet our response time goal of .700 ms per transaction.
Table 9-1  Excessive objects performance index

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual response percentage</td>
<td>95%</td>
<td>18%</td>
</tr>
<tr>
<td>Performance index</td>
<td>1.00</td>
<td>****</td>
</tr>
</tbody>
</table>

We repeated this display several times, however, the performance degradation did not vary significantly.

**Throughput**

The next step is to check the throughput and compare it to our baseline runs. The transaction rate has dropped from approximately 50 - 60 per second to approximately 25 per second, see Table 9-2. This represents more than a 50% reduction in throughput.

Table 9-2  Excessive objects throughput

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction ended rate</td>
<td>50</td>
<td>60</td>
<td>24.72</td>
</tr>
</tbody>
</table>

**Response time**

We check the response times and compare them to our baseline runs, see Table 9-3. We find that the response time has increased from approximately .125 -.175 ms to approximately 2.600 ms. This represents an increase of almost 20 times the response time or 2000%, which is quite substantial.

Table 9-3  Excessive objects response time

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait time</td>
<td>.080</td>
<td>.100</td>
<td>2.137</td>
</tr>
<tr>
<td>Execution time</td>
<td>.045</td>
<td>.075</td>
<td>.482</td>
</tr>
<tr>
<td>Actual time</td>
<td>.125</td>
<td>.175</td>
<td>2.619</td>
</tr>
</tbody>
</table>

**Conclusions**

At this point, we can quantify the impact of the performance problem at the application server level. We know that the throughput has dropped and the average response time has increased.

We also see that the wait time has increased from an average of approximately .090 ms to over 2.1 seconds. This is more than a 2000% increase and may be more responsible for the increase than the execution time, which has an
approximate increase of 800%. Therefore, we are not only taking longer to execute a transaction, but also waiting substantially longer during the course of the transaction.

In the case of this scenario, the end user will notice the problem immediately because going from two-tenths of a second to more than 2.5 seconds is a substantial increase.

9.2.2 Response time distribution

We check the response time distribution screen (see Figure 9-2). This screen shows the response time and throughput information, and also allows you to graphically see how the transactions are distributed according to response time.

![Figure 9-2  RMF response time distribution for scenario 2](image)

**Response time and throughput**

The response time and throughput in this display (see Table 9-4) are the same as for the sysplex summary display. This is to be expected since the displays are done within seconds of each other and our sample has not changed.

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait time</td>
<td>.080</td>
<td>.100</td>
<td>2.137</td>
<td></td>
</tr>
<tr>
<td>Execution time</td>
<td>.045</td>
<td>.075</td>
<td>.482</td>
<td></td>
</tr>
</tbody>
</table>
Response time distribution
When we check the response time distribution, we can see a dramatic shift to the right. Almost all of our transactions seem to be impacted and very few have met the response time goal.

9.2.3 Work Manager delays

We check the Work Manager delays screen (see Figure 9-3). This screen in addition to the response time and throughput information allows you to see whether there are any system delays.

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual time</td>
<td>.125</td>
<td>.175</td>
<td>2.619</td>
</tr>
<tr>
<td>Transaction rate</td>
<td>50</td>
<td>60</td>
<td>24.72</td>
</tr>
</tbody>
</table>

Throughput and response time
This display shows the same time period as the two preceding displays (see Figure 9-1 and Figure 9-2) and contains the same information for throughput and response time.
Note that the transaction rate (TRX) shown in this display is 6008. This is the number of transactions that were processed in the last 100 seconds, which equals a rate of 60.08 per second. Table 9-5 lists the scenario 2 Work Manager delays throughput and response time.

<table>
<thead>
<tr>
<th>Table 9-5</th>
<th>Excessive objects: Work Manager delays throughput and response time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>Low</td>
</tr>
<tr>
<td>For ---- transaction</td>
<td>5000</td>
</tr>
<tr>
<td>Average response time</td>
<td>.125</td>
</tr>
<tr>
<td>Average execution time</td>
<td>.045</td>
</tr>
</tbody>
</table>

Also note that the wait information is not displayed on this screen. You can determine this by subtracting the execution time from the response time.

Delayed by information

9.2.4 System information

Next we check the system information screen (see Figure 9-4). This screen shows the service class, with the high response time and lower throughput (TRANS/SEC). It also shows a very small amount of processor delay and no other delays. The average CPU utilization to the entire logical partition (LPAR) is very low and therefore not the cause of any problem.

![Figure 9-4  RMF system information for scenario 2](image-url)
Throughput
From this point we can surmise that the change in performance is not system-related and we must check server or application problems. This requires changing to a different tool to complete the analysis. Table 9-6 lists the low throughput in the system information screen.

Table 9-6  Excessive objects: System information throughput

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction/SEC</td>
<td>50</td>
<td>60</td>
<td>24.72</td>
</tr>
</tbody>
</table>

Response time
Table 9-7 lists the high response time in the system information screen.

Table 9-7  Excessive objects: System information response time

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response time</td>
<td>.125</td>
<td>.175</td>
<td>2.62</td>
</tr>
</tbody>
</table>
9.3 RMF reports for scenario 2

In addition to checking the RMF displays, it is also important to see the RMF report for each of the five-minute intervals during the time period that we run each scenario.

9.3.1 Workload activity report

The workload activity report (see Figure 9-5) shows our goal, which is to have 95% of the transactions completed in less than .700 ms.

![Figure 9-5](image)

**Performance index**

As shown in the five-minute period, this problem is very severe. The performance index is ****, which indicates that it is greater than 4. The actual percentage of transactions meeting the response time goal during this time period is 50.1%, see Table 9-8.

| Table 9-8  Excessive objects: Workload activity performance index |
|-------------|----------------|----------------|
| Baseline   | Normal | This scenario |
| Actual response percentage | 95% | 50.1% |
| Performance index | 1.00 | **** |
Response time distribution
Response time distribution shows a significant shift, where almost 40% of all transactions are running 1 second or longer.

9.3.2 Workload activity transactions

The transaction section of the workload activity report (see Figure 9-6) shows a total volume of 11128 transactions, a throughput of 37.06 tps, transaction queue time of 689ms, execution time of 296ms, and a total response time of 985ms. The total number of transactions is slightly lower than the baseline of approximately 15000 transactions, and throughput is down to almost 25% from the baseline of 50 tps.

![Figure 9-6 Workload activity report: Transactions for scenario 2](image)

**Throughput**

The transaction section of the workload activity report shows a total volume of 11128 transactions and a throughput of 37.06 tps, as shown in Table 9-9.

<table>
<thead>
<tr>
<th>Table 9-9 Excessive objects: Workload activity throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baseline</strong></td>
</tr>
<tr>
<td>Transaction ended</td>
</tr>
<tr>
<td>END/S</td>
</tr>
</tbody>
</table>
Response time
The different times reported for the transactions in this scenario are significantly longer, see Table 9-10. However, the data does not suggest a cause.

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual time</td>
<td>.125</td>
<td>.175</td>
<td>.985</td>
</tr>
<tr>
<td>Execution time</td>
<td>.045</td>
<td>.075</td>
<td>.296</td>
</tr>
<tr>
<td>Queued time</td>
<td>.080</td>
<td>.100</td>
<td>.689</td>
</tr>
</tbody>
</table>

9.3.3 System summary report
The summary report for the same time period shows CPU utilization of 79.6% and minimal system paging, as shown in Figure 9-7.

Table 9-11 shows the CPU busy and demand paging output.

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU busy</td>
<td>35%</td>
<td>45%</td>
</tr>
<tr>
<td>Demand paging</td>
<td>.30</td>
<td>3.0</td>
</tr>
</tbody>
</table>
9.4 TPV for scenario 2

In this scenario, differences are apparent in memory usage. These differences are only apparent when compared to the baseline. When we check them with no baseline reference point, these differences may not appear significant.

9.4.1 Java virtual machine runtime

Figure 9-8 shows the Java virtual machine (JVM) heap display from scenario 2. This chart displays both free memory and used memory. It also shows that when garbage collection occurs, the available memory does not drop as low as it does in the baseline run. This is because there are more objects in memory. In this case garbage collection occurs in intervals slightly less than a minute.

![Figure 9-8 TPV JVM heap display for scenario 2](image-url)
9.5 OMEGAMON for scenario 2

We use OMEGAMON to dig deeper into the application running on the WebSphere Application Server for z/OS.

9.5.1 Workload Manager (WLM) service class resources

The initial indication of a performance problem on the OMEGAMON monitor comes from a trigger set on the performance index. In this case, we start with one of the screens that shows the performance index and the address space bottlenecks display, see Figure 9-9.

![Figure 9-9  OMEGAMON WLM service class resources for scenario 2](image-url)
9.5.2 Address space bottlenecks

The address space bottleneck screen (see Figure 9-10) shows the WASPM service class with a performance index of 4.1, which is highlighted in red.

![Address space bottlenecks diagram](image)

Figure 9-10 OMEGAMON address space bottlenecks for scenario 2

After reviewing the other MVS system statistics, we cannot identify any system constraints or delays that can account for the increase in response time. Therefore, we look at the servant region.
9.5.3 Servant regions

The servant regions display (see Figure 9-11) shows the two server instances as expected. The Hypertext Transfer Protocol (HTTP) sessions graph indicates approximately 49 HTTP sessions per server. This is approximately the same as in our baseline runs, and indicates that the overall volume of request coming into the servers has not changed significantly.

Figure 9-11 OMEGAMON servant regions for scenario 2
9.5.4 Garbage collection activity

We check the garbage collection activity display to check whether any change has occurred in memory usage. Figure 9-12 shows an increase in the garbage collection rate of one server, doubling from one per minute in the baseline to almost two per minute.

![Figure 9-12  OMEGAMON garbage collection activity for scenario 2](image-url)
9.5.5 Garbage collection history

By drilling down on the server in question, we see in the display (see Figure 9-13) that garbage collection is not only higher, but is also still increasing and the real percentage of time spent in garbage collection is also increasing.

Another useful display is the heap usage display, which shows the objects contained within the heap. Drilling down through these displays can identify the most common objects in the heap.

![Figure 9-13 OMEGAMON garbage collection detail for scenario 2](image)

From this point, we begin to look at specific application components to try and isolate the specific area where the response time increase is most likely to have come from.
9.5.6 Web application

The Web application display (see Figure 9-14) shows average servlet response time, in this case, approximately 66 ms. Although a bit higher than our baseline, it is not nearly enough to account for the 200 ms increase in the overall transaction.

Next, we examine the Enterprise JavaBeans (EJB) components.

Figure 9-14 OMEGAMON Web application for scenario 2
9.5.7 EJB containers

The first display of the EJB container (see Figure 9-15) shows the bean instantiation rate and total invocations for the past 60 seconds.

![Figure 9-15  EJB containers for scenario 2](image-url)
9.5.8 Total method invocations

Drilling down to the next display (see Figure 9-16) shows a graph of the EJBs in the container and the number of method invocations for each. The table at the bottom shows each bean, which is sorted by response time. The top row shows a bean with a 201 ms average response time.

![Figure 9-16 Total method invocations for scenario 2](image-url)
9.5.9 Ten worst average response times

When you select the first row in Figure 9-16 and further drill down, this displays the methods for the selected bean. Figure 9-17 graphically shows the ten methods with the longest response times.

From these displays the analysis switches to the application profile, if available, to check for changes in the number of calls to a specific method or the relative length of time spent in a specific method. Without a pre-existing profile, the next step is to review the longest running methods and compare for the most common object seen in the heap usage analysis.
9.6 WSAM for scenario 2

In the WebSphere Studio Application Monitor (WSAM), the first indication of the change in performance is based on the average response time. We set a trigger level for this on the server statistics overview screen. As one of the possible default pages, we start at the server overview screen.

9.6.1 Server overview

The server overview scene (see Figure 9-18) shows five basic graphs for the selected server: Throughput, average response time, number of sessions, JVM CPU utilization, and JVM memory utilization. We see that response time is consistently over 250 ms and throughput is down below 1200 transactions per minute or 20 tps. Both of these show significantly worse performance than our baseline. However, the total number of sessions continues to be consistent with the baseline number, showing no increase in incoming volume. JVM CPU is also slightly higher than the baseline.

Next, we check the response time distribution.
9.6.2 Response time distribution

The WLM service class response time distribution detail scene (see Figure 9-19) shows the same distribution change as shown in both the RMF reports and RMF displays. The majority of the transactions show response times well over the service class goal.

![Response time distribution for scenario 2](image)

From this point, we use a series of memory analysis screens to view a number of metrics.
9.6.3 Memory analysis

First, we look at JVM heap activity and live sessions, see Figure 9-20. This analysis compares the impact of incoming volume to heap activity. In this case, volume is consistent, however, the heap utilization is increases slightly after each garbage collection.
We compare the garbage collection rate with throughput, see Figure 9-21. Throughput is reasonably stable through all of the garbage collection events. However, note that the garbage collection rate is twice the baseline rate.

Figure 9-21  Garbage collection versus number of requests for scenario 2
Finally, we compare garbage collection time to response time, see Figure 9-22. Again, both the garbage collection time and response time are reasonably stable, though response time is up from the baseline.

From the basic memory analysis, it does not appear that memory utilization has any significant impact on the change in response time or throughput.

From this point, the analysis will continue checking the application components, and using the Web application summary and EJB summary as in the previous scenario. For more details, see 8.6.3, “Servlet summary” on page 125, and 8.6.4, “EJB method summary” on page 126.
Scenario 3: Serialization
10.1 Overview

In this scenario, we take the basic Trader application and modify a couple of the methods, which causes the application to serialize processing on specific objects. This is a fairly common programming problem that impacts performance by reducing the effectiveness of the multi-threading capability of the server. The difference in this scenario is that the impact of the change is purposely kept to a minimum so that it is not as obvious as the previous scenarios.

10.2 RMF displays for scenario 3

When we start a scenario, we use the Resource Measurement Facility (RMF) Monitor III and display the sysplex summary screen. This highlights any service class that does not meet its performance goal.

10.2.1 Sysplex summary

The sysplex summary shows that our goal is to have 95% of the transactions completed in less than .700 ms, see Figure 10-1.

![RMF sysplex summary for scenario 3](image)
Performance index
In this case, the performance index is 1.10, which indicates only a slight performance degradation, see Table 10-1. We still complete 95% of the transactions within our response time goal of .700 ms per transaction.

Table 10-1  Serialized object performance index

<table>
<thead>
<tr>
<th>Baseline</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual response percentage</td>
<td>95%</td>
<td>95%</td>
</tr>
<tr>
<td>Performance index</td>
<td>1.00</td>
<td>1.10</td>
</tr>
</tbody>
</table>

We performed multiple displays and noticed that this 10% performance degradation was fairly consistent.

Throughput
The next step is to check the throughput and compare it to our baseline runs. The transaction rate is normal during this display, see Table 10-2. There is no reduction in throughput during the course of this display time period. Therefore, this is not the cause of the change in the performance index.

Table 10-2  Serialized object throughput

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction ended rate</td>
<td>50</td>
<td>60</td>
<td>60.08</td>
</tr>
</tbody>
</table>

Response time
We check the response times and compare them to our baseline runs. We find that the overall response time of .170 is on the high side of what we consider normal for our environment. A closer look indicates that the wait time has increased to slightly more than what we consider normal, see Table 10-3. This may be a partial cause of the problem.

Table 10-3  Serialized object response time

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait time</td>
<td>.080</td>
<td>.100</td>
<td>.103</td>
</tr>
<tr>
<td>Execution time</td>
<td>.045</td>
<td>.075</td>
<td>.69</td>
</tr>
<tr>
<td>Actual time</td>
<td>.125</td>
<td>.175</td>
<td>.170</td>
</tr>
</tbody>
</table>
**Conclusions**

At this point, we can quantify the impact of the performance problem at the application server level. We know that the throughput is still consistent with our baseline, but the average response time has increased slightly. According to Figure 10-1, we have a minor performance problem, which marginally seems to be impacting our response time.

We continue monitoring until it is determined that this was not just a single bad period, but is a consistent increase in the wait times. Unless you continually monitor the performance, you may easily overlook a slight increase such as the one in this example. It is important to notice that there is a problem when there is an increase in the performance index.

### 10.2.2 Response time distribution

We check the response time distribution screen (see Figure 10-2). This screen shows the response time and throughput information and also allows you to graphically see how the transactions are distributed according to response time.

![Figure 10-2 RMF response time distribution for scenario 3](image)

*Figure 10-2  RMF response time distribution for scenario 3*
Response time and throughput
The response time and throughput in this display (see Table 10-4) are the same as for the sysplex summary display. This is to be expected since the displays are done within seconds of each other and our sample has not changed.

Table 10-4  Serialized object response time and throughput

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait time</td>
<td></td>
<td>.080</td>
<td>.100</td>
<td>.103</td>
</tr>
<tr>
<td>Execute time</td>
<td></td>
<td>.045</td>
<td>.075</td>
<td>.068</td>
</tr>
<tr>
<td>Actual time</td>
<td></td>
<td>.125</td>
<td>.175</td>
<td>.170</td>
</tr>
<tr>
<td>Transaction rate</td>
<td>50</td>
<td>60</td>
<td>60.08</td>
<td></td>
</tr>
</tbody>
</table>

Response time distribution
The response time distribution shows normal operation, therefore, we do not use this graphical display.

10.2.3 Work Manager delays
We check the Work Manager delays screen (see Figure 10-3). This screen shows the response time and throughput information, and allows you to see whether there are any system delays.

Figure 10-3  RMF Work Manager delays for scenario 3
Response time
Table 10-5 lists the response time output in the Work Manager delays screen.

Table 10-5  Serialized object: Work Manager delays response time

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual</td>
<td></td>
<td>95%</td>
<td>95%</td>
</tr>
<tr>
<td>Average response time</td>
<td>.125</td>
<td>.200</td>
<td>.170</td>
</tr>
<tr>
<td>For ---- TRX</td>
<td>5000</td>
<td>6000</td>
<td>6008</td>
</tr>
<tr>
<td>Average execution time</td>
<td>.045</td>
<td>.080</td>
<td>.068</td>
</tr>
</tbody>
</table>

10.2.4 System information

Finally, we see the system information display (see Figure 10-5) to check for system influences on the transaction response times. The central processing unit (CPU) utilization and delays show no significant system constraint for this period.

Figure 10-4   RMF system information for scenario 3
Response time
Table 10-6 lists the response time and throughput output in the system information screen.

Table 10-6  Serialized object: System information response time and throughput

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response time</td>
<td>.125</td>
<td>.175</td>
<td>.170</td>
<td></td>
</tr>
<tr>
<td>Transaction/SEC</td>
<td>50</td>
<td>60</td>
<td>60.08</td>
<td></td>
</tr>
</tbody>
</table>

Conclusions
In general, we did not find any system level cause for the change in performance behavior. Therefore, we switch tools to check for server or application causes.

10.3 RMF reports for scenario 3

In addition to checking the RMF displays, it is also important to see the RMF report for each of the five-minute intervals during the time period that we run each scenario.

10.3.1 Workload activity report

The workload activity report (see Figure 10-5) shows our goal, which is to 95% of the transactions completed in less than .700 ms.

Figure 10-5  RMF workload activity report: Performance goal for scenario 3
Performance index

The performance index for each of the five-minute intervals during the course of this scenario is consistently above one. In this case, it is at 1.2, with 94.4% of the transactions meeting the response time goal, see Table 10-7.

Table 10-7  Serialized object: Workload activity performance index

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual response percentage</td>
<td>95%</td>
<td>94.4%</td>
<td></td>
</tr>
<tr>
<td>Performance index</td>
<td>1.00</td>
<td>1.20</td>
<td></td>
</tr>
</tbody>
</table>

This varies only slightly from the RMF system summary display where the performance index is 1.1, see Table 10-1.

Response time distribution

The response time distribution did not change noticeably from the baseline.

10.3.2 Workload activity transactions

The basic response time metrics for the scenario show an overall response time of 181 ms with an execution time of 69 ms and a queued time of 112 ms, see Figure 10-6. All of these values are slightly higher than the baseline values. The throughput of 59.04 tps for this five-minute period still falls in the range seen during the baseline. However, other periods for this run show a slight decrease.
Transactions
Table 10-8 lists the transactions in the workload activity report screen.

| Table 10-8   Serialized object: Workload activity transactions |
|---|---|---|---|
| Baseline | Low | High - normal | This scenario |
| Transaction ended | 15000 | 16000 | 17712 |
| END/S | 50 | 60 | 59.04 |

10.4 TPV for scenario 3
We did not use TPV this scenario.

10.5 OMEGAMON for scenario 3
We did not use OMEGAMON for this scenario.

10.6 WSAM for scenario 3
The response time trigger on the server statistics overview display is the first indicator of a performance change.

10.6.1 Server overview
In Figure 10-7, the server statistics overview indicates that the response time trigger level has been exceeded.

Figure 10-7   WSAM server statistics overview for scenario 3
10.6.2 Server summary

The Server summary display (see Figure 10-8) shows a couple of spikes in response time along with corresponding drops in throughput over a period of 30 minutes.
10.6.3 Memory analysis

A review of the heap statistics using the memory analysis display (see Figure 10-9) shows no significant change in the heap usage or garbage collection. The Java virtual machine (JVM) heap usage is consistent with the baseline run. However, the number of requests is somewhat erratic.

![Memory analysis display](image)

These server metrics are inconclusive in determining a cause for the performance change. Therefore, further review of the Web application and EJB container components indicated a change in specific EJB methods as in previous scenarios.
Scenario 4: Java virtual machine heap size
11.1 Overview

In this scenario, we use the basic Trader application without any changes. But we modify the server configuration to increase the Java virtual machine (JVM) heap size from 256 M to 768 M. If you decrease the frequency of Java garbage collection by increasing the heap size, it can cause unwanted system memory paging when enough real memory is not available.

11.2 RMF displays for scenario 4

When we start a scenario, we use the Resource Management Facility (RMF) Monitor III and display the sysplex summary screen. This highlights any service class that does not meet its performance goal.

11.2.1 Sysplex summary

The sysplex summary shows that our goal is to have 95% of the transactions completed in less than 0.700 ms, see Figure 11-1.
Performance index
In this display, the performance index is 1.20, which indicates only a slight performance degradation, see Table 11-1. We still complete 94% of the transactions within our response time goal of .700 ms per transaction.

Table 11-1  Heap size performance index

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual response percentage</td>
<td>95%</td>
<td>94%</td>
<td></td>
</tr>
<tr>
<td>Performance index</td>
<td>1.00</td>
<td>1.20</td>
<td></td>
</tr>
</tbody>
</table>

This case is slightly different from the previous scenarios. This is because the performance index varies from normal to more than 4.00 and then back to normal during the course of several displays.

Throughput
The next step is to check the throughput and compare it to our baseline runs. The transaction rate has dropped slightly from approximately 50 - 60 per second to approximately 49 per second, see Table 11-2. This represents a very slight reduction in throughput and shows that we were in a fairly normal period when this display was performed.

Table 11-2  Heap size throughput

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction ended rate</td>
<td>50</td>
<td>60</td>
<td>49.23</td>
<td></td>
</tr>
</tbody>
</table>

Response time
We check the response times and compare them to our baseline runs. We find that the total response time has increased from approximately .125 ms - .175 ms to approximately .275 ms, see Table 11-3. This represents an increase in response time of approximately 30%, with majority of the problem caused by the increased wait time.

Table 11-3  Heap size response time

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait time</td>
<td>.080</td>
<td>.100</td>
<td></td>
<td>.184</td>
</tr>
<tr>
<td>Execution time</td>
<td>.045</td>
<td>.075</td>
<td></td>
<td>.091</td>
</tr>
<tr>
<td>Actual time</td>
<td>.125</td>
<td>.175</td>
<td></td>
<td>.275</td>
</tr>
</tbody>
</table>
Though the preceding display (see Table 11-2) shows only a slight problem, the problem increased and decreased periodically. We had several displays that showed no degradation, while others showed a performance index of greater than 4.00.

**Conclusions**

At this point, we can quantify the impact of the performance problem at the application server level. We know that the throughput has dropped and then recovered at the same time that the average response time increased and decreased.

There seems to be a periodic pattern to the times when we have poor performance. This can be indicative of something external to the transaction itself that is causing the slowdown.

### 11.2.2 Job delays

The Job delays screen (see Figure 11-2) shows the delays associated with the servant region CZSR01AS. The primary delay for both servant address spaces is storage waits of 71% and 62%.

![Figure 11-2  RMF job delays for scenario 4](image)

**Conclusions**

Job delays due to paging of private area storage points to a memory management problem.
11.2.3 Storage frames

This RMF screen (see Figure 11-3) shows the number of storage frames used by each address space and the page-in rate for each. This screen shows the three servant address spaces BZSR01AS as the highest users of real memory on the system and the page-in rate for each.

![RMF Storage Frames](image)

*Figure 11-3  RMF storage frames for scenario 4*

**Conclusions**

Because the address spaces are the highest users of real memory, it indicates that a memory management problem is the underlying cause of the increase in overall response times.
11.3 RMF reports for scenario 4

In addition to checking the RMF displays, it is also important to see the RMF report for each of the five-minute intervals during the time period that we run each scenario.

11.3.1 Workload activity report

The workload activity report (see Figure 11-4) shows our goal, which is to have 95% of the transactions completed in less than .700 ms.

![Figure 11-4 RMF workload activity report: Performance goal for scenario 4](image)

**Performance index**

Because only 80% of the transactions met their response time goal during this time period, the performance index shows ****. This means that the performance index is greater than 4, see Table 11-4.

<table>
<thead>
<tr>
<th>Baseline</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual response percentage</td>
<td>95%</td>
<td>80.1%</td>
</tr>
<tr>
<td>Performance index</td>
<td>1.00</td>
<td>****</td>
</tr>
</tbody>
</table>

We check each of the five-minute intervals to see whether we can discern a time pattern to the periodic poor performance.
Response time distribution
This particular five-minute time period, the response time distribution has shifted, with 12% of the total transactions taking more than 2.8 seconds to complete.

11.3.2 Workload activity transactions

The transaction section of the workload activity report shows the total throughput has dropped to 22.62 tps, with an average response time of 5131 ms, execution time of 390 ms, and queued time of 4741 ms. All these values are much higher than the baseline. Figure 11-5 shows the transaction details.

![Figure 11-5 Workload activity report: Transactions for scenario 4](image)

Throughput
The throughput is measured from the baseline of 50 - 60 tps, see Table 11-5.

<table>
<thead>
<tr>
<th>Table 11-5 Increased heap size: Workload activity throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baseline</strong></td>
</tr>
<tr>
<td>Transaction ended</td>
</tr>
<tr>
<td>END/S</td>
</tr>
</tbody>
</table>

Response time
Table 11-6 shows the response time in the workload activity report screen.

<table>
<thead>
<tr>
<th>Table 11-6 Increased heap size: Workload activity response time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baseline</strong></td>
</tr>
<tr>
<td>Actual time</td>
</tr>
</tbody>
</table>
### 11.3.3 Workload activity class WASPC

For the most part, we focus on the workload activity report for the service class WASPM. In this scenario, we also look at the report class WASPC, which is the class used for the server control regions. The report shows the page-in rates and storage for our servers, each significantly higher than on the baseline run, see Figure 11-6.

**Figure 11-6  Workload activity report: Class WASPC for scenario 4**

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution time</td>
<td>.045</td>
<td>.075</td>
<td>.390</td>
<td></td>
</tr>
<tr>
<td>Queued time</td>
<td>.080</td>
<td>.100</td>
<td>4.741</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Execution time</th>
<th>Queued time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>.045</td>
<td>.080</td>
</tr>
<tr>
<td>High - normal</td>
<td>.075</td>
<td>.100</td>
</tr>
<tr>
<td>This scenario</td>
<td>.390</td>
<td>4.741</td>
</tr>
</tbody>
</table>

**Page-in rates**
11.3.4 Summary report

The summary report (see Figure 11-7) for the same period shows the overall swap rate and demand paging for the system. In this case too the numbers are significantly higher.

![RMF SUMMARY REPORT]

**Figure 11-7** Summary report for scenario 4
11.4 TPV for scenario 4

We repeated the baseline run with the JVM size increased to 512 MB, and then again increased this to 712 MB. The difference from the baseline is evident in the Tivoli Performance Viewer (TPV) JVM runtime display, in the patterns of garbage collection.

11.4.1 JVM runtime

The chart in Figure 11-8 is from the 512 MB heap size. It shows that some garbage collections are taking much longer than in the baseline. The intervals between the collections are also much more erratic.

![Figure 11-8  JVM runtime memory utilization: 512 MB heap](image)
When the heap size is increased to 712 MB, the garbage collections are much less frequent, see Figure 11-9. When they do occur, they cause hung transactions, that is, the Enterprise JavaBeans (EJBs) cannot load while the heap memory is locked for collection. Therefore, the collection takes a long period of time.

Figure 11-9  JVM runtime memory utilization: 712 MB heap
11.5 OMEGAMON for scenario 4

We use OMEGAMON to dig deeper into the application running on the WebSphere Application Server for z/OS.

11.5.1 Workload Manager (WLM) service class resources

The initial indication of a performance problem on the OMEGAMON monitor comes from a trigger set on the performance index. In this case, we start with one of the screens that shows the performance index and the WLM service class resources display, see Figure 11-10.

![Figure 11-10 OMEGAMON WLM service class data for scenario 4](image-url)
11.5.2 Address space bottlenecks

With OMEGAMON, we use the address space bottlenecks display to check the impact of the system paging, see Figure 11-11. OMEGAMON graphically shows the storage wait for our servant regions CZSR01AS, and our server control region CZSR01A.

Figure 11-11  OMEGAMON address space bottlenecks data for scenario 4
11.5.3 Servant regions

Figure 11-12 shows the servant region screen.

Figure 11-12 OMEGAMON servant region memory size data for scenario 4
11.5.4 Garbage collection activity

The OMEGAMON garbage collection activity display (see Figure 11-13) shows another impact of the larger heap size. Garbage collection occurs less frequently. However, when it does occur, it has a greater impact. In this case, we see that one garbage collection event is close to 12% in real percentage.

Figure 11-13  Run 19: OMEGAMON heap usage history for scenario 4
11.6 WSAM for scenario 4

WebSphere Studio Application Monitor (WSAM) has the capability to capture system paging information. We can use this as one of the metrics in a memory analysis display. However, in our installation of WSAM, the data capture agent for system paging was not active and we could not review system paging. Other memory analysis displays did show some of the impact of the large JVM heap size.

11.6.1 Memory analysis

The first memory analysis display shows the number of garbage collection events against the average response time. Figure 11-14 shows that garbage collection only happened once in over a ten-minute period.

![Figure 11-14  WSAM memory analysis: Number of garbage collection versus average response time](image-url)
In the second display (see Figure 11-15), we compare the length of the garbage collection events with the throughput or number of requests completed. This display shows that garbage collections are very long in duration and have a direct impact on the throughput of the server when they occur.

Figure 11-15  WSAM memory analysis: Garbage collection time versus requests
Scenario 5: Increased volume
12.1 Overview

This scenario is the simplest performance change, where the incoming request delivery rate increases. For this scenario, we use the unmodified Trader application. However, we increase the number of concurrent clients from 100 to 200 and, accordingly, increase the number of requests to the server.

Although volume increases are fairly easy to identify, once identified you need to understand the impact that either a momentary spike or long-term volume growth may have on other areas of performance.

Changes in volume may affect memory utilization and garbage collection, which may require additional heap size tuning. If additional volume causes queued time to increase, additional threads may be necessary. If multiple servant regions are enabled, the additional volume may trigger the Workload Manager (WLM) to start a new servant region instance.

12.2 RMF displays for scenario 5

When we start a scenario, we use the Resource Measurement Facility (RMF) Monitor III and display the sysplex summary screen. This highlights any service class that does not meet its performance goal.

12.2.1 Sysplex summary

The sysplex summary shows that our goal is to have 95% of the transactions completed in less than .700 ms, see Figure 12-1.

![Figure 12-1 RMF sysplex summary for scenario 5](image-url)
Performance index
In this case, the performance index is 1.30, which indicates a slight performance degradation, see Table 12-1. We still complete 93% of the transactions within our response time goal of .700 ms per transaction.

<table>
<thead>
<tr>
<th>Baseline</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual response percentage</td>
<td>95%</td>
<td>93%</td>
</tr>
<tr>
<td>Performance index</td>
<td>1.00</td>
<td>1.30</td>
</tr>
</tbody>
</table>

We repeated this display several times and found the performance degradation did not vary significantly.

Throughput
The next step is to check the throughput and compare it to our baseline runs. The transaction rate has increased from approximately 50 - 60 per second to approximately 75 per second, see Table 12-2. This represents an approximate increase of 30% in throughput.

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction ended rate</td>
<td>50</td>
<td>60</td>
<td>75.30</td>
</tr>
</tbody>
</table>

Response time
We check the response times and compared them to our baseline runs. The response time has increased from approximately .125 ms - .175 ms to almost .275 ms, see Table 12-3. This represents an increase in response time of approximately 30%. A closer look shows that the problem seems to be related to the wait time, which has more than doubled.

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Low</th>
<th>High - normal</th>
<th>This scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait time</td>
<td>.080</td>
<td>.100</td>
<td>.204</td>
</tr>
<tr>
<td>Execution time</td>
<td>.045</td>
<td>.075</td>
<td>.066</td>
</tr>
<tr>
<td>Actual time</td>
<td>.125</td>
<td>.175</td>
<td>.275</td>
</tr>
</tbody>
</table>
Conclusions
At this point, we can quantify the impact of the performance problem at the application server level. We know that the throughput has increased and that the average response time has also increased. Even though our response time has risen, our transaction throughput has also increased. Because we normally run with only 100 users at a fairly steady state, the increase is possibly related to an increased number of users.

12.2.2 Work Manager delays
We check the Work Manager delays screen (see Figure 12-2). This screen shows the response time and throughput information, and allows you to see whether there are any system delays.

![Figure 12-2  RMF Work Manager delays for scenario 5](image)

Throughput
We see a significant increase of approximately 2000 transactions more than our baseline throughput.

Response time
However, the average response times are significantly longer.

Conclusions
The degradation in response is most likely caused by the increased volume of transactions.
12.3 RMF reports for scenario 5

In addition to checking the RMF displays, it is also important to see the RMF report for each of the five-minute intervals during the time period that we run each scenario.

12.3.1 Workload activity report

The workload activity report (see Figure 12-3) shows our goal, which is to have 95% of the transactions completed in less than .700 ms.

![Figure 12-3   RMF workload activity report: Performance Index for scenario 5](image)

Performance index

The performance index for this run shows an increase to 1.2, see Table 12-4. This is because only 94.3% of our transactions meet the response time goal.

| Table 12-4  Increased volume: Workload activity performance index |
|-------------|-----------------|-----------------|
| Baseline   | High - normal   | This scenario   |
| Actual response percentage | 95% | 94.3% |
| Performance index | 1.00 | 1.20 |

Response time distribution

The response time distribution for this five-minute period shows only a slight change from the baseline.
12.3.2 Workload activity transactions

The transaction section of the workload activity report shows an increase in volume from the number of transactions that ended in the five-minute period, 23383 compared to approximately 15000 during the baseline run. In addition, the throughput is up to 77.91 tps, an increase of about 20 tps. However, the transaction response time also increased with an average response time of 196 ms, a somewhat higher execution time of 62 ms, and a significantly high queued time of 134 ms. Figure 12-4 shows this output.

![Figure 12-4 Workload activity: Transactions for scenario 5](image)
12.4 TPV for scenario 5

When the volume of transactions is increased, it is easy to detect the difference using Tivoli Performance Viewer (TPV). You can see this when the settings are changed to show deltas (Settings → View Data As → Change in Value). Under this setting, TPV displays the number of servlets (Enterprise JavaBeans (EJBs) or Web applications, depending on what is viewed) invoked since the last recording interval.

12.4.1 Servlets

In this mode (see Figure 12-5), we can see that the total requests received under the TraderDBEAR servlet is usually more than 2000 in 10-second intervals (in other words, more than 200/second). The number of concurrent requests and the response times have both risen when compared with the baseline.

![Figure 12-5 TPV servlet detail data for scenario 5](image-url)
12.5 OMEGAMON for scenario 5

We use OMEGAMON to dig deeper into the application running on the WebSphere Application Server for z/OS.

12.5.1 Workload Manager (WLM) service class

The initial indication of a performance problem on the OMEGAMON monitor comes from a trigger set on the performance index. In this workload display, we start with one of the screens that shows the performance index and the address space bottlenecks display, see Figure 12-6. Once again OMEGAMON triggers the WLM service class display when the performance index exceeds 1.

![Figure 12-6 OMEGAMON WLM service class for scenario 5](image)
12.5.2 Servant regions

When we check the servant regions display (see Figure 12-7), it shows the dramatic increase in Hypertext Transfer Protocol (HTTP) sessions.

![Figure 12-7 OMEGAMON WLM servant regions for scenario 5](image-url)
12.6 WebSphere Studio Application Monitor (WSAM) for scenario 5

12.6.1 Server overview

In the server or group overview displays, the increase in throughput is shown graphically along with response time. Figure 12-8 shows the jump when the number of clients is changed from 100 to 200.

![Figure 12-8  WSAM server overview for scenario 5](image-url)
12.6.2 Memory analysis

To determine the impact of the additional volume on memory utilization, we look at the memory analysis display, see Figure 12-9. When we check the number of garbage collections per minute and the heap size following garbage collection, we can see that the garbage collection occurs more frequently than in the baseline, but is still less than once per minute. Garbage collection also appears to be recovering more than half of the heap at each collection.

Figure 12-9  WSAM memory analysis: Number of garbage collection versus heap size
In this second display (see Figure 12-10), we compare the total garbage collection time against requests to check whether garbage collection impacts throughput. We see that total garbage collection times are, for the most part, short and consistent with minimal impact on completed requests.

Figure 12-10  WSAM memory analysis: Number of garbage collection versus number of requests
Scenario 6: Memory leak
13.1 Overview

Memory leaks have long been a major problem with Java. In this scenario, we deliberately introduce a memory leak into the Trader application to monitor this characteristic. Most memory leaks that make it into a production server are very small leaks that take a long period of time to develop into a problem with a memory shortage. However, in the case of this scenario, we create a very fast leak in order to show the pattern over a short run. In addition, we reduce the number of concurrent clients to 50 to get consistent results.

13.2 Resource Measurement Facility (RMF) displays for scenario 6

With the reduced load and a Trader application that is for the most part unmodified, the performance of the transactions is equal to that of the baseline run. Throughout this run response times are normally below 50 ms and the performance index is at .5. Because the target of this scenario is garbage collection monitoring, we have omitted the RMF displays.

13.3 RMF reports for scenario 6

For the same reasons as with the RMF displays, we have omitted the RMF reports for this scenario.
13.4 Garbage collection trace for scenario 6

When diagnosing performance problems, it is useful to check the garbage collection (GC) trace. This trace can alert you to a number of different problems. In this scenario, the trace shows the standard symptom of a memory leak.

Note the decrease in the amount of free space left, eventually this will lead to a problem, see Figure 13-1. In this case, the memory leak is large enough to be spotted quickly. In many cases, the leak is much more insidious and may take days to manifest itself.

![Garbage collection trace](image.png)

Figure 13-1 Decreasing free space
13.5 TPV for scenario 6

The memory leak has a distinctive pattern that we see in the Java virtual machine (JVM) runtime memory display in Tivoli Performance Viewer (TPV).

13.5.1 JVM runtime

Each time garbage collection is performed, the amount of free memory decreases, see Figure 13-2. Eventually, the amount of free memory reaches zero, at which point the server address space will terminate.
13.6 OMEGAMON for scenario 6

You can use OMEGAMON to dig deeper into the application running on the WebSphere Application Server for z/OS.

**Suspected memory leaks**

OMEGAMON has a specific suspected memory leaks display to monitor the JVM heap for continuous growth, see Figure 13-3. In addition it provides additional information to help identify the source of the leak. Due to a technical problem with our installation of OMEGAMON, we are not able use this display during this scenario.

![Figure 13-3 OMEGAMON suspected memory leaks](image-url)
13.7 WebSphere Studio Application Monitor (WSAM) for scenario 6

13.7.1 Number of garbage collections versus JVM heap size

WSAM memory analysis displays quickly identify a memory leak pattern. Figure 13-4 shows a comparison of the number of garbage collections to heap size. In this case, the garbage collection shows the decreasing amount of available memory over time.

Figure 13-4   WSAM memory analysis: Number of garbage collections versus heap size
13.7.2 Garbage collection time versus heap size

When we compare total garbage collection time to heap size, we see an increase in garbage collection time as available heap decreases, as shown in Figure 13-5.
Appendixes
Monitoring environment

This appendix provides a general overview of the basic infrastructure necessary to support the monitoring scenarios described previously in this book.
Overview

Previously, we determined that all of the scenarios that we chose for this book can be created using a single WebSphere 5.1 Base Application Server Node installation. We decided against setting up a Network Deployment Node in order to keep things as simple as possible.

Consequently, we installed and configured a WebSphere Base Application Server Node (AZSR01A) on a single z990 logical partition (LPAR) running the z/OS R1.5 operating system within the International Technical Support Organization (ITSO) Sysplex.

Next, it seemed feasible to us to install the OMEGAMON and WSAM performance monitoring products on this z990 LPAR to monitor both the z/OS platform and the WebSphere 5.1 middleware. The intent was to limit the amount and complexity of the infrastructure as much as possible to allow the team to focus on the setup and running of the scenarios.

Unfortunately, we found out that the releases we were running of the OMEGAMON and WSAM products are not compatible when they are both set up to monitor the same WebSphere Base Application Server. There is no conflict regarding the z/OS data collections, therefore, we could still maintain a single z/OS image. But we need to have two WebSphere Application Servers installed, one being monitored by OMEGAMON and the other by WSAM.

At this point, since both products were monitoring the AZSR01A server, the quickest solution was to remove the WSAM monitoring from the AZSR01A server and to create an additional BZSR01A server in support of the WSAM monitoring. This leaves the AZSR01A in support of the OMEGAMON.

As things turned out this was not a good idea. The WSAM product, when installed and configured into a WebSphere Base Application Server Node, cannot be uninstalled. This restriction applies to Base Application Server Nodes only. To perform an uninstall in this environment will result in a permanent failure, whenever you attempt to start the server. Unaware of this restriction, we uninstalled WSAM from the AZSR01A server and that was the end of that server as a functioning entity.

The solution at that point was to create two new Base Application Servers, the BZSR01A server for monitoring by WSAM and the CZSR01A server for monitoring by OMEGAMON. This proved to be the final configuration, a single z990 LPAR running z/OS with two WebSphere Base Application Servers installed.
Appendix A. Monitoring environment

Infrastructure release levels

WebSphere information

- Build number W510004
- Build date of 28 July 2004
- Java 2 Runtime Edition (J2RE) 1.3.2; build level cm1420-20040626

OMEGAMON (Candle) information

- OMEGAMON XE for WebSphere Application Server for OS/390
  - Version 130
  - PTF QWW1303 applied
- OMEGAMON XE for OS/390, Version 130
WebSphere Studio Application Monitor (WSAM) (Cyanea) information

Version 3.1

Tivoli Performance Viewer (TPV)

- WebSphere Application Server: BASE Version 5.1.0
- Build Number b0344.02
- Build Date 11/4/03
Testing environment

This appendix describes the WebSphere Workload Simulator (WSWS), which we used to drive our scenario workloads.
Baseline setup

The setup for our baseline runs is as follows:

- Called the application TraderDBEAR from WSWS using a peak load of 100 concurrent clients.
- Installed TraderDBEAR on a WebSphere Application Server for z/OS server with two servant regions for a total of 12 execution threads. Each servant region had a 256 M heap.
- The Workload Manager (WLM) service class for our workload was WASPM and the goal was set at 95% at 700 ms.

WebSphere Studio Workload Simulator

The WebSphere Studio Workload Simulator (WSWS) allows you to simulate thousands of users simultaneously accessing the WebSphere Application Server for z/OS.

A series of transactions can be captured and formatted into a script. That script can then be fed to the desired number of clients for execution.

For more information on WSWS, refer to Overview of WebSphere Studio Application Monitor and Workload Simulator, SG24-6073.

WSWS usage

We used WSWS to drive our test systems with 100 simultaneous clients. This provided us enough data to measure performance without straining the capabilities of our test systems.

Besides obtaining baseline information about the performance of a production environment application by measuring the actual environment, we believe that it is also worthwhile to run a controlled baseline test that can be repeated.

WSWS allows you to capture scripts that can be rerun so that you can easily compare the current data with historical data.

Our scripts were also simplistic. We were mainly concerned with how our tools could detect the root causes of each scenario and not establishing a realistic quality control environment by using complex scripts and multiple driving WSWS systems.
Runtime Monitor

The Runtime Monitor screen allowed us to track the course of each scenario as we ran it, as shown in Figure B-1.

![WSWS Runtime Monitor screen]

*Figure B-1  WSWS Runtime Monitor screen*
The Monitor screen is divided into three parts: A graphical section (see Figure B-2), an overview section, and a console messages section.

**WSWS graphs**

![WSWS Monitor graphs](image)

*Figure B-2  WSWS Monitor graphs*

We used the Graphical section to tell at a glance the relative performance of each of our scenario runs.

**Changing the number of Clients**

We typically ran with 100 clients on our single base application server, see Figure B-3.

![Changing the number of Clients](image)

*Figure B-3  Changing the number of Clients*
WSWS console messages

Figure B-4  WSWS console log

WSWS Script

Trader Application

Our application, Trader, was very simplistic. It allows you to request a quote and either buy or sell the stock of a limited number of companies. For more information on Trader, refer to WebSphere for z/OS V6 Connectivity Handbook, SG24-7064.
Index

B
BBOC_HTTP_TRANSACTION_CLASS  41

G
garbage collection  42

I
installation  71

J
JVM heap size  37
JVM_ENABLE_VERBOSE_GC  35
JVM_HEAPSIZE  37

M
memory leak  35, 37

O
Overview  71

R
Redbooks Web site  218
  Contact us  xv
response time
  objective  42

S
Server Region address space classification  42
Server Region enclave classification  40

T
transaction
  class  41

V
verbose GC trace  35

W
WLM queues  40
Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 218. Note that some of the documents referenced here may be available in softcopy only.

- *Overview of WebSphere Studio Application Monitor and Workload Simulator*, SG24-6073
- *WebSphere for z/OS V5 Problem Determination*, SG24-6880
- *WebSphere for z/OS V6 Connectivity Handbook*, SG24-7064

Other publications

These publications are also relevant as further information sources:

- *z/OS V1R2.0 MVS Programming: Workload Management Services*, SA22-7619
- *WebSphere Application Server for z/OS: V5.1 Performance Tuning and Monitoring*, SA22-7963-02
- *WebSphere Studio Application Monitor Installation and Customization Guide V2.1.4*, SC31-6312
- *WebSphere Studio Application Monitor Operations Guide V2.1.4*, SC31-6313
- *WebSphere Studio Application Monitor User’s Guide*, SC31-6314
- *z/OS V1R7.0 RMF User’s Guide*, SC33-7990
- *z/OS V1R7.0 RMF Report Analysis*, SC33-7991
- *z/OS V1R7.0 RMF Performance Management Guide*, SC33-7992
- *Installation & Configuration of Candle Products on OS/390 and Z/OS*, GC32-9214
Online resources

These Web sites and URLs are also relevant as further information sources:

- WebSphere Application Server Infocenter
  http://publib.boulder.ibm.com/infocenter/ws51help/index.jsp
- WebSphere V5 for z/OS download site
- Technical Sales Library
  http://www.ibm.com/support/techdocs/
- Resource Measurement Facility (RMF) home page

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
To determine the spine width of a book, you divide the PPI into the number of pages in the book. An example is a 250 page book using Plainfield opaque 50# smooth which has a PPI of 526. Divided 250 by 526 which equals a spine width of 0.4752". In this case, you would use the 0.5" spine.

Now select the Spine width for the book and hide the others:

Special>Conditional Text>Show/Hide>Spine Size (-->Hide:)>Set

Move the changed Conditional text settings to all files in your book file with the spine.fm still open and File>Import>Format Conditional Text Settings (ONLY!) to the book files.
To determine the spine width of a book, you divide the paper PPI into the number of pages in the book. An example is a 250 page book using Plainfield opaque 50#, which has a PPI of 526. Divided 250 by 526 which equals a spine width of .4752". In this case, you would use the .5" spine width.
# Abbreviations and acronyms

This abbreviations and acronyms file is optional. Use this file by adding names and descriptions to it. Sort these names: highlight rows > Table > Sort > Sort By: Column 1 > Sort or optionally add names and descriptions to the Index file instead of this file by indexing the first use of an abbreviation or acronym: highlight text > Special > Marker > Index > New Marker

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abbreviation1</td>
<td>Description1</td>
</tr>
<tr>
<td>abbreviation2</td>
<td>Description2</td>
</tr>
<tr>
<td>IBM</td>
<td>International Business</td>
</tr>
<tr>
<td></td>
<td>Machines Corporation</td>
</tr>
<tr>
<td>ITSO</td>
<td>International Technical</td>
</tr>
<tr>
<td></td>
<td>Support Organization</td>
</tr>
<tr>
<td>abbreviation3</td>
<td>Description3</td>
</tr>
<tr>
<td>abbreviation4</td>
<td>Description4</td>
</tr>
</tbody>
</table>
Monitor and troubleshoot production performance of WebSphere on z/OS

This IBM Redbook was written for IBM zSeries users, performance analysts, system administrators and system engineers who need a comprehensive understanding of IBM WebSphere on z/OS performance management in order to ensure the successful deployment of e-business applications. Performance monitoring and system tuning in a production environment is a vast and complex topic. Hypes and claims of performance and scalability are confusing and often misleading. This redbook helps you understand WebSphere Application Server V5.1 performance factors, and how you can monitor your system and application performance, response time, and resource utilization. It provides practical hints and tips on various real-life factors that influence the performance of applications in production on WebSphere on z/OS.

The book provides a general introduction to WebSphere runtime and discusses the key performance factors in a z/OS production environment. Beyond general recommendations, we describe a performance troubleshooting approach. Scenarios are used to explain how to narrow down to the source of the problem. Interpretation of data and rules of thumb are provided.