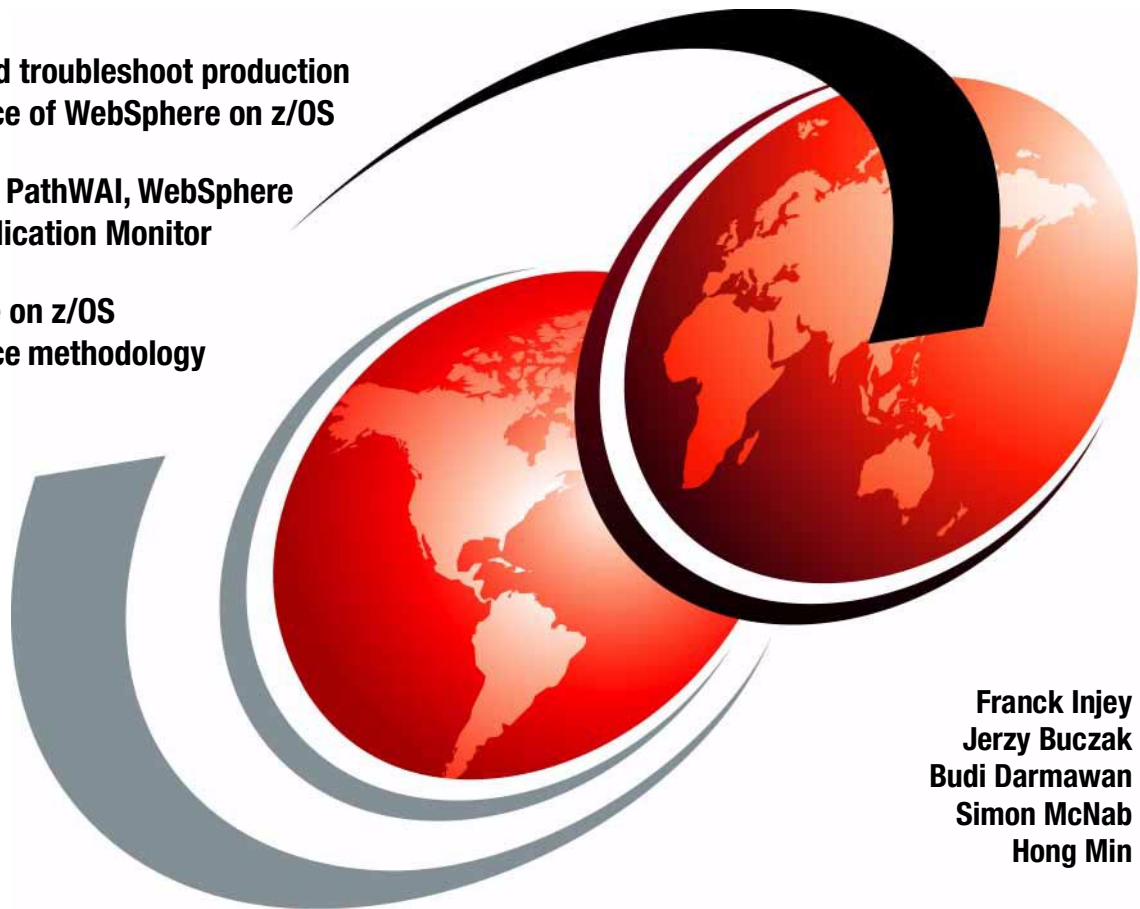


Monitoring WebSphere Application Performance on z/OS

Monitor and troubleshoot production performance of WebSphere on z/OS

Introscope, PathWAI, WebSphere Studio Application Monitor

WebSphere on z/OS performance methodology



Franck Injey
Jerzy Buczak
Budi Darmawan
Simon McNab
Hong Min



International Technical Support Organization

**Monitoring WebSphere Application Performance on
z/OS**

April 2003

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

First Edition (April 2003)

This edition applies to WebSphere Application Server V4.0.1 for z/OS and OS/390 at Service Level 4, program number 5655-F31 for use with z/OS Version 1 or OS/390 Version 2 Release 10. This document created or updated on April 10, 2003.

© Copyright International Business Machines Corporation 2003. All rights reserved.

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

Contents

Notices	vii
Trademarks	viii
Preface	ix
The team that wrote this redbook	x
Become a published author	xi
Comments welcome	xii
Part 1. Walking the WebSphere performance path	1
Chapter 1. WebSphere runtime on z/OS	3
1.1 zSeries hardware and z/OS	4
1.1.1 Central Processors and logical partitions	4
1.1.2 Parallel Sysplex®	5
1.1.3 Address spaces and tasks	5
1.1.4 z/OS components	6
1.2 The WebSphere programming model	7
1.2.1 Java overview	7
1.2.2 J2EE	8
1.3 Application server model	11
1.3.1 Regions and instances	12
1.3.2 Servers and nodes	13
1.4 System administration model	15
1.5 Putting it together: a typical customer installation	16
1.6 Performance components	18
1.6.1 The TCP/IP network	19
1.6.2 zSeries server	19
1.6.3 z/OS	19
1.6.4 The application	23
Chapter 2. WebSphere and z/OS, walking the performance path	25
2.1 Introduction to performance and terminology	26
2.1.1 Setting your performance expectations	28
2.1.2 Performance management	29
2.1.3 How to know that you have a performance problem	30
2.1.4 What to do about a performance problem	31
2.2 Workload Manager controls	33
2.3 Gathering WebSphere performance information	39
2.3.1 SMF records	39

2.3.2	RMF reports	39
2.3.3	DB2 SMF records	54
2.3.4	WebSphere SMF records	57
2.3.5	Garbage Collection (GC) trace	67
2.4	Establishing the diagnosis	70
2.4.1	Overview	70
2.4.2	Initial diagnostics	72
2.4.3	Where does it hurt?	78
2.4.4	Check for memory problem	82
2.4.5	The delay pain	83
2.4.6	The CPU pain	85
Chapter 3.	The ITSO test environment	87
3.1	Hardware and software configuration	88
3.1.1	The sysplex configuration	88
3.1.2	Network access	92
3.1.3	ITSO test workloads	96
3.1.4	WebSphere Studio Workload Simulator	98
3.2	Examples of performance problems	99
3.3	Performance monitoring tools	101
Part 2.	WebSphere performance tools	107
Chapter 4.	Introscope	109
4.1	Introscope	110
4.1.1	Introscope major components	111
4.1.2	Monitoring WebSphere on z/OS	112
4.1.3	Enterprise Manager	116
4.1.4	Workstation	116
4.1.5	Introscope performance and monitoring methodology	123
4.1.6	ITSO configuration	123
4.2	Examples	126
4.2.1	Example 4: CICS	126
4.2.2	Example 6: No DB2 Index	129
4.2.3	Example 10: Too Much Logging	134
4.2.4	Example 3: Memory Leak	136
4.2.5	Example 1: Identify Bad User	140
4.2.6	Example 7: Transaction Hang	144
4.2.7	Example 8: Static Pages	145
4.2.8	Example 10: Increased WebSphere Activity	148
4.2.9	Example 11: Prioritizing Problems	153
Chapter 5.	PathWAI solutions for WebSphere	157
5.1	PathWAI solutions	158

5.2 OMEGAMON XE performance monitors	158
5.2.1 OMEGAMON XE architecture	158
5.2.2 Monitoring WebSphere Application Server	162
5.2.3 Monitoring the WebSphere environment.	166
5.3 PathWAI configuration at ITSO	170
5.4 Analyzing the ITSO examples	173
5.4.1 Example 1 - Identify a DB2 delay in the application path	173
5.4.2 Example 3 - Detect a memory leak	181
5.4.3 Example 4 - Identify a CICS TS response time problem.	188
5.4.4 Example 6 - Isolate a DB2 problem	194
5.4.5 Example 7 - Transaction hang or time-out	200
5.4.6 Example 8 - Static pages serving	207
5.4.7 Example 9 - Increased WebSphere activity	212
5.4.8 Example 10 - Identify a method called with high frequency	217
5.4.9 Example 11 - Detecting multiple concurrent problems	220
Chapter 6. WebSphere Studio Application Monitor	225
6.1 What WebSphere Studio Application Monitor is	226
6.2 How WebSphere Studio Application Monitor works	228
6.2.1 WebSphere Studio Application Monitor architecture.	228
6.2.2 WebSphere Studio Application Monitor data collection	229
6.2.3 WSAM Application Monitor	231
6.2.4 WSAM Monitor Console	233
6.3 Performance methodology	233
6.4 ITSO configuration	236
6.5 Running the examples.	237
6.5.1 Example 1	237
6.5.2 Example 3	243
6.5.3 Example 4	245
6.5.4 Example 6	253
6.5.5 Example 7	260
6.5.6 Example 8	264
6.5.7 Example 9	268
6.5.8 Example 10	271
Appendix A. Java and J2EE details	279
A.1 Java class loading.	280
A.2 Java runtime execution.	280
A.3 Java memory and garbage collection	281
A.4 J2EE application flow	286
A.5 J2EE application structure	288
Appendix B. Configuration files	291
B.1 HTTP Server definitions	292

B.2 z/OS TCP/IP definitions	298
Related publications	301
IBM Redbooks	301
Other resources	301
Referenced Web sites	302
How to get IBM Redbooks	302
IBM Redbooks collections	302
Index	303

Notices

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

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

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:


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

Trademarks

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

eServer™
z/OS™
z/VM™
zSeries™
AIX®
AS/400®
CICS®
CT™
DB2®
™

IBM®
IMS™
Language Environment®
Lotus®
MQSeries®
MVS™
Notes®
OS/390®
OS/400®
Parallel Sysplex®

PC 300®
Redbooks™
Redbooks (logo) ™
RACF®
RMF™
S/390®
WebSphere®
1-2-3®

The following terms are trademarks of other companies:

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

Candle Management Server, AF/REMOTE, CandleNet Portal, OMEGAMON, Alert Adapter, CMS, eBusiness at the speed of light are trademarks or registered trademarks of Candle Corporation.

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

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

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

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

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

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

Preface

This IBM Redbook 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 V4.0.1 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 or OS/390®.

The book is divided into two parts:

- ▶ Part 1 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. Examples are given to explain how to narrow down to the source of the problem. Interpretation of data and rules of thumb are provided.
- ▶ Part 2 expands on performance monitoring products available for WebSphere on z/OS that will help detect and identify performance problems.
 - Candle Corp. PathWAI™ Dashboard for WebSphere Infrastructure
 - IBM WebSphere Studio Application Monitor
 - Wily Technology Introscope®

For each product, we describe the relevant methodology and show through typical real-life examples, what to look for in determining where the performance bottleneck is.

Important: The purpose of this publication is to document performance tools and techniques. Although it contains many measurement examples, they were run in a non-controlled environment. Each measurement is only an example to illustrate a given function or technique; the results shown are not intended to represent precisely what can be achieved in other environments.

Furthermore, the hardware configuration did not remain constant over the duration of the project. Comparison of measurement results across different sets of examples would lead to erroneous conclusions and should not be attempted.

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.

Franck Injey is a Project Leader at the International Technical Support Organization, Poughkeepsie. He has 30 years of experience working on S/390® hardware and system performance. Before joining the ITSO, Franck was a Consulting I/T Architect in France.

Jerzy Buczak is a Consulting I/T Specialist working for WebSphere zSeries Software Services in Research Triangle Park, North Carolina. Prior to that, he was responsible for OS/390-based networking at the ITSO in Raleigh, NC, with eight redbooks to his name. Jerzy holds a degree in mathematics from Cambridge University, England, and has over 20 years of experience in mainframe networking.

Budi Darmawan is a Consulting IT Specialist at the International Technical Support Organization, Austin Center. He writes extensively and teaches IBM classes worldwide on all areas of systems management and database administration. Before joining the ITSO three years ago, Budi worked in IBM Global Services Integrated Technology Services in IBM Indonesia as a Technical lead and Solution Architect. His areas of expertise include Tivoli solutions, database administration, business intelligence, and OS/390 administration.

Simon McNab is an I/T Specialist working for IBM Global Services in Portsmouth, UK. He specializes in WebSphere Application Server for z/OS and also has experience in WebSphere on AIX® and Linux for zSeries. He has worked as a Systems Programmer for z/OS, VM/VSE, CICS® and a number of database products. He holds a degree in Computer Science from Pembroke

College, Cambridge, in the United Kingdom and has 15 years of mainframe experience.

Hong Min is a software engineer at the IBM Design Center for e-business on demand(TM), Poughkeepsie, USA. She has more than five years of experience in zSeries e-business technical support, customer application design, and prototyping. She holds an MS degree in Computer Science from Drexel University, Philadelphia, PA. Her areas of expertise include z/OS, Java, J2EE, and WebSphere.

Special thanks to Bob St. John of zSeries Performance, IBM Poughkeepsie, for his extensive and valuable technical support of this project.

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

David Bennin, Richard Conway
International Technical Support Organization, Poughkeepsie

Bart Steegmans
International Technical Support Organization, Santa Teresa

Mark Addleman
Wily Technology Inc.

Lindsay Farmer, Peter Kingsley, Girish Kulkarni, Warren Macek, Michael McDonald, Clyde Richardson, Don Zeunert
Candle Corporation

Robert Lam, Richard Mackler, Arthur Tsang
Cyanea Systems

Nin Lei, Worldwide Strategy and Scalability Center
IBM Poughkeepsie

Dave Cohen, WebSphere for z/OS Development
IBM Poughkeepsie

Arti Kadakia, WebSphere for z/OS Performance
IBM Poughkeepsie

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 Internet note 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

Walking the WebSphere performance path



WebSphere runtime on z/OS

In this chapter we describe how the runtime components of WebSphere fit together and run together in a z/OS sysplex. WebSphere is an extensive collection of applications, so we confine ourselves to those components necessary to run (not develop) applications in production. Those same components are the ones of interest to us in monitoring performance.

We cover the following topics:

- ▶ What is z/OS?
- ▶ What does a WebSphere application look like?
- ▶ How do the WebSphere application servers work under z/OS?
- ▶ What does it all look like when it is put together?
- ▶ What can affect its performance?

1.1 zSeries hardware and z/OS

We include this brief introduction to z/OS for the benefit of those readers unfamiliar with the mainframe environment. WebSphere on z/OS is very different from WebSphere on distributed platforms, and much of that difference is due to the unique nature of the zSeries architecture.

The zSeries architecture is optimized for a mixed workload. Therefore, you will find the WebSphere servers sharing their mainframe with databases, transaction processing, development, batch jobs, and almost everything else. z/OS ensures that each piece of work is allocated the resources and the priority it needs to fulfill the installation's service objectives.

In performance terms, this means that poor response time in WebSphere could be due to excessive resource usage by another application. And what is more, this could be “working as designed”. If the business has determined that Web applications should *not* have the highest priority, then at peak times WebSphere transactions could receive poor service. This is not a performance issue, this is a business issue.

1.1.1 Central Processors and logical partitions

Each physical zSeries server comes with one or more Processing Units (PUs), some storage, and a Channel Subsystem that communicates with the outside world. Each PU can perform one of a number of functions depending on the microcode loaded into it. The PUs that run the operating system are called Central Processors (CPs). However, there is not a one-to-one mapping between instances of the operating system and the CPs. Rather, each copy of the operating system runs in its own *logical partition* (LPAR). An LPAR has assigned to it a certain proportion of the total computing power, a certain amount of storage, and a certain number of channels. Moreover, these proportions can be dynamically adjusted as the workload changes.

When we talk about work running on a mainframe, we talk in terms of LPARs because that is how the mainframe is seen by an application: One LPAR = one system image = one interface between the application and the server. But you need to remember that one LPAR could be using as many as 16 CPs, or only half a CP.

Many operating systems, such as z/OS, z/VM™ and Linux run in zSeries LPARs. In this book we are only concerned with z/OS.

1.1.2 Parallel Sysplex®

To achieve high availability for a z/OS application, you need that application to run in multiple LPARs, ideally distributed between multiple physical servers. However, if you also want optimum performance, you need to ensure very close coordination between those LPARs. These two principles give rise to the concept of the *sysplex*. A sysplex comprises multiple z/OS LPARs (spread across one or more physical servers) that have three things in common:

- ▶ Shared disk space, containing (at the very least) the files (data sets in z/OS parlance) that define the way the sysplex LPARs cooperate
- ▶ A means of communication, called the *cross system coupling facility (XCF)*, that allows the sysplex LPARs to keep in touch and up to date with all interesting events
- ▶ A Sysplex Timer that keeps the physical servers' clocks in synchronization. The timer is not required if the whole sysplex is in the same physical server, as there is only one clock.

The coupling facility is a very high-speed shared storage area that can be used by z/OS instances for holding critical data. Using the coupling facility allows in-storage data to survive the failure of a z/OS instance, since any other instance can retrieve the data and continue to process it.

The coupling facility is in fact just another LPAR, running a special operating system optimized for just the one purpose. Coupling facilities are connected to z/OS LPARs via high-speed connections. In a production environment there are usually at least two coupling facilities for redundancy.

A typical customer environment might include several physical zSeries servers, containing a production sysplex, a development sysplex, and a test sysplex. The production sysplex might comprise:

- ▶ A z/OS LPAR using two CPs in server A
- ▶ A z/OS LPAR using three CPs in server B
- ▶ A coupling facility in server C
- ▶ A z/OS LPAR using one CP in server D
- ▶ A coupling facility in server D

The development and test sysplexes would probably have less computing power, and only one coupling facility each.

1.1.3 Address spaces and tasks

Turning to a single z/OS in a single LPAR, we now take a look at how work gets run within this environment.

When you run a job or start an application under z/OS (for example, a WebSphere application server), the operating system creates an *address space*. This is simply a piece of virtual storage that is assigned to the application for the duration of its existence, and the application runs within it. Up to 2 Gb of virtual storage is available for each address space, at least until 64-bit addressing is implemented. As well as user applications, many of z/OS's own system tasks run in their own address spaces.

The address space is the higher of the two layers of work management in z/OS. The lower unit is the *task*. The individual task is what gets dispatched by z/OS when it is ready to do work and has the highest priority of all the ready tasks. When an address space is started, the application that was invoked is assigned a task, and can work under the auspices of that task. If it needs to perform multiple units of work concurrently, it can request z/OS to create new tasks and assign them to the units of work. For example, if the WebSphere application server is handling several client requests at the same time, you would expect to see several tasks running in its address space.

1.1.4 z/OS components

Many of the major functions of z/OS run within address spaces rather than within the supervisory kernel. They have special authority and privileges. As far as WebSphere performance is concerned, two of the most important z/OS components are Workload Manager and UNIX System Services.

Workload Manager

One of the most important performance-related components of any z/OS sysplex is the workload manager (WLM). An instance of it runs on each z/OS LPAR, and together they build a picture of the workload currently running in the sysplex.

The installation defines to WLM the performance policies (goals) that apply to various items of work. WLM uses these goals, together with its knowledge of the running workload, to:

- ▶ Manage workload distribution and balancing, which includes scheduling new address spaces to handle increasing workload
- ▶ Distribute resources to competing workloads

UNIX System Services

To ease the portability of applications from other platforms to zSeries, z/OS includes a component called UNIX System Services (USS). It behaves as an operating system within an operating system, and provides:

- ▶ UNIX APIs
- ▶ A hierarchical file system (HFS) similar to that used on UNIX
- ▶ A UNIX-like command shell

Applications running under z/OS can make use of the traditional z/OS APIs, or the UNIX APIs, or both.

In UNIX, applications run as *processes*; each process can comprise multiple *threads*. In z/OS, these are generally mapped to address spaces and tasks (TCBs), respectively.

WebSphere makes extensive use of the USS programming interfaces.

1.2 The WebSphere programming model

WebSphere Application Server Version 4 follows the specifications laid down in the Java 2 Platform Enterprise Edition (J2EE), Version 1.2. The J2EE V1.2 specification is itself based on the Java language programming platform, and it defines many aspects of enterprise Java applications including:

- ▶ The internal design of enterprise Java applications
- ▶ The mechanisms used by the application to communicate to other systems
- ▶ The process for deploying code into a server environment

1.2.1 Java overview

Java is an object-oriented programming language developed by Sun Microsystems Inc. It has the look-and-feel of C++, but is easier to use than C++ . It includes a comprehensive set of APIs that ranges from desktop GUI to database access to make a programmer's life easier. Since Java was introduced in 1995, it has been used extensively to build simple and complex applications.

Java programming model

In order to understand the J2EE programming model, it is important to understand the basics of the Java programming model. Java enforces object-oriented programming. Each type of object is represented by a construct called *class*, which has methods and variables. Classes can have subclasses, and together they build the class hierarchy. Another important piece is Interface, a device that unrelated objects use to interact with each other. Interfaces only have methods, which can be implemented differently by different classes. See A.2, "Java runtime execution" on page 280.

For code clarity and manageability reasons, Java has the concept of a *package*. A package groups logically related classes in the same way that a file system directory groups logically related files. There is no runtime performance impact with regard to packages.

In general, Java class files are packaged into one or more Java Archive (jar) files. This is a file format that is based on the popular ZIP file format for aggregating many files into one.

Java runtime execution

Java classes are not stored as object code in the sense that the hardware understands it. They are stored as bytecode to be interpreted by the *Java Virtual Machine* (JVM). Although this uses more runtime resources than traditional compilation, its great advantage is portability. Although the JVM is always platform-specific, the classes it runs are platform-independent. JVM also has Just-In-Time (JIT) compile function, which dynamically compiles bytecode into execution code to improve performance

But in this book we are not dealing simply with Java applications. We are talking specifically about z/OS WebSphere applications, which add another degree of sophistication. And WebSphere adheres to the J2EE standard for developing, deploying, and running enterprise applications. So what is J2EE?

1.2.2 J2EE

The J2EE specification builds on the Java specification and provides the facilities intended to make building and managing enterprise applications easier. J2EE defines three basic elements of any enterprise application: components, containers and connectors. WebSphere provides a set of logical resources to increase the performance of the various containers and connectors. See “Components, containers, and connectors” on page 9 for more information.

J2EE also provides a standard set of services, available through APIs:

- ▶ Java Naming and Directory Interface (JNDI), which enables components to locate objects they require.
- ▶ Java DataBase Connectivity (DBC), which lets components manipulate existing data from relational and other databases.
- ▶ Remote Method Invocation - Internet Inter Orb Protocol (RMI-IIOP), which provides a communication method for components to talk to other applications
- ▶ Java Message Service (JMS) provides a means for applications to exchange messages asynchronously with (for example) MQ.
- ▶ Java Transaction API (JTA) allows applications to manage their own transactions if the services provided by the container are not to their taste.
- ▶ JavaMail provides the ability to send e-mail from within a Java application. JavaMail includes the JavaBeans Activation Framework (JAF), an API used to handle the data in e-mail messages.

- ▶ Java Connector Architecture (JCA) is an emerging standard for connector access.

Components, containers, and connectors

Components are Java code built by application developers that follow particular guidelines. J2EE defines many different kinds of components, each intended to fulfill specific requirements. Examples of components include HTML, servlets, JSPs and Enterprise Java Beans, and GUI client-like applets.

Containers manage the life cycle of J2EE components, and every J2EE component is managed by a container. Containers are responsible for creating and destroying components, pooling components, and dispatching requests from external sources to the appropriate component. Examples of containers include the Web container, EJB container, and application client container.

Connectors provide facilities for Java applications to interact with systems external to WebSphere such as DB2®, CICS, IMS™, etc. Note that the external systems need not be data stores; the external system could be another WebSphere server region or a third-party application such as Siebel or legacy applications. In order to maintain transactional integrity between these external systems, J2EE defines the Java Transaction API (JTA).

J2EE components

J2EE defines two server-side application components for e-business transactions:

- ▶ Servlets and JSPs
- ▶ Enterprise Java Beans (EJBs)

Servlets and JSPs are managed by the Web container and are responsible for interpreting HTTP requests and composing responses, normally in the form of HTML. A JSP is a template for a Web page; it supplies the presentation content for a page. The code for a JSP looks much more like HTML with Java code embedded in it. The Web container compiles the JSP into a servlet when it is first invoked. From an execution standpoint, there is no difference between a JSP and a servlet.

An EJB is a set of Java classes that defines a piece of the business logic used in a Web application. Not surprisingly, EJBs are managed by the EJB container. There are three types of EJBs:

Entity beans

An entity bean implements an object view of an entity stored in the database, or it can implement an existing enterprise application. There are two types of entity

beans: Container-Managed Persistence (CMP), and Bean-Managed Persistence (BMP).

Session beans

A session bean is a non-persistent object responsible for the business logic of a particular user transaction. It manages the interactions of entity and other session beans, accesses resources, and generally performs tasks on behalf of the client. There are two types of session beans: stateful session beans, which are dedicated to a client and hold the conversational-state for the client; and stateless session beans, which does not hold conversational-state and can be shared among clients.

Message beans

A message-driven bean is an asynchronous message consumer. It is invoked by the container as a result of the arrival of a JMS message.

Application designers and developers can choose between BMP and CMP when implementing entity beans. In some cases, BMP beans have to be used due to the entity it represents, for example, a legacy application. From a performance point of view, CMP is a good choice because of all the work being done in WebSphere Application Server to improve CMP performance. BMP can perform as well if the application developers are performance-conscious and write very efficient code in the application.

J2EE containers

The Web container is responsible for receiving HTTP requests and dispatching them to the appropriate JSP or servlet. As described above, there is no execution difference between JSPs and servlets. The Web container can be configured to create and initialize servlets either at startup or on demand when the particular servlet is invoked for the first time. servlets are normally multi-threaded, so the Web container will only create a single instance of any particular servlet.

The EJB container receives requests from external clients and dispatches them to the appropriate EJB for service. The protocol used to communicate to the EJB container can vary depending on the type of service requested. The most common protocol is RMI, but CORBA IIOP is also used.

The programmer's view of a J2EE server is shown in Figure 1-1 on page 11.

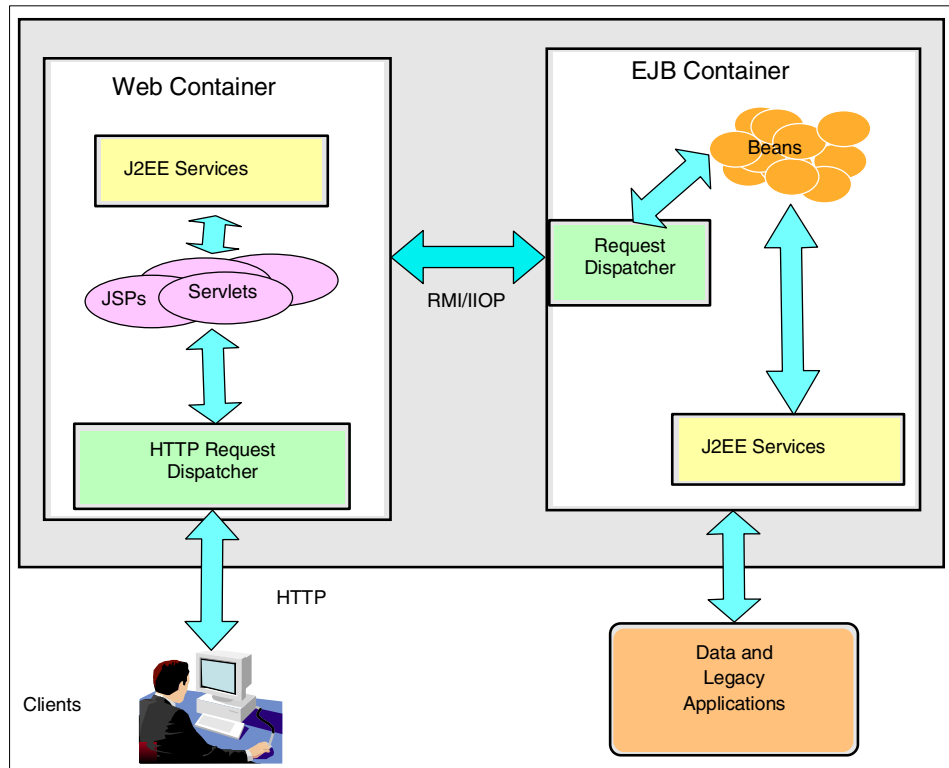


Figure 1-1 J2EE server structure

In terms of assembly and deployment, a J2EE application is packaged using the Java Archive (JAR) file format into a file with an ear (Enterprise Archive) file name extension. An ear file can contain Web components in Web Archive (war) files, EJB beans in ejb-jar files, dependent library jar files, and the J2EE deployment descriptor stored with the name META-INF/application.xml in the ear file. Not all components are necessary, but at least a war or ejb-jar file and the deployment descriptor should be included.

1.3 Application server model

So now that we have some J2EE applications set up in the correct way, we need an environment within z/OS that will execute these applications.

1.3.1 Regions and instances

A J2EE application container, which includes both Web container and EJB container, runs *in a process* in UNIX terminology. This translates to an address space in z/OS language, called the *server region*. Since we require some measure of availability and scalability, we create multiple server regions and manage the distribution of work to them via another address space called the *control region*. The combination of a control region and its managed server regions is called a *server instance*. Figure 1-2 illustrates this.

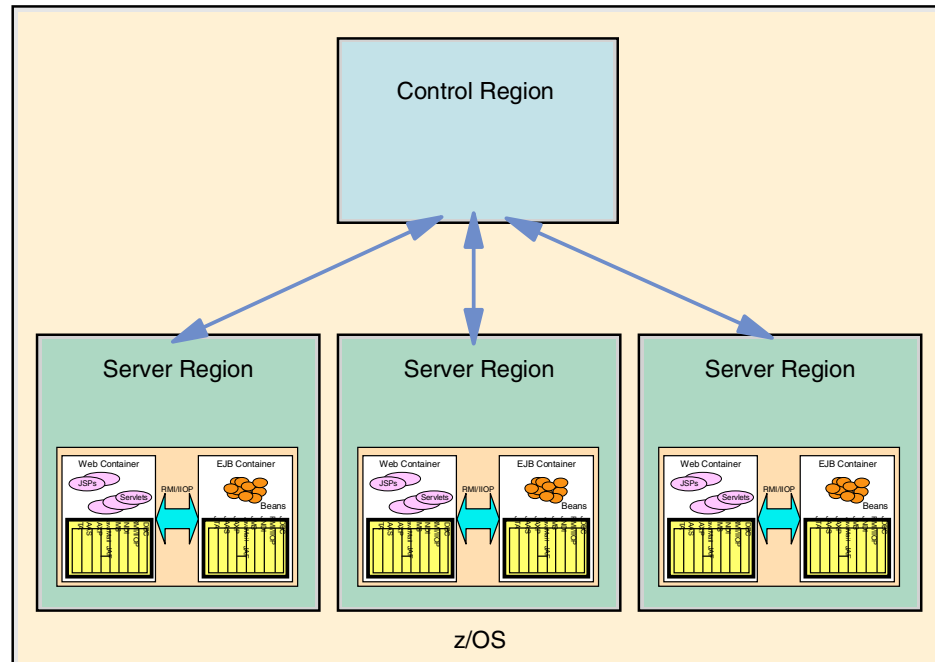


Figure 1-2 WebSphere Application server instance

The control region is the end point of the communication (TCP connection) from the client. Its job is to distribute requests to the server regions. It places the requests on a WLM queue, from which WLM takes them and gives them to the server region that it deems best able to meet the performance goal. WLM is also able to start new server regions if the existing ones are near their capacity limit.

Within each server region (address space), many client requests may be handled concurrently. Each request runs as a *thread* in UNIX terminology, or a subtask in z/OS terminology.

1.3.2 Servers and nodes

Now we have a control region plus a group of server regions. This still does not constitute high availability. The next stage is to clone the server instances, so that a number of control regions accept requests from the network, passing them on to the appropriate server regions within their own domains. The group of server instances is called a *server*. It is the entire server that presents itself to the clients, so that clients perceive a complete collection of control regions and server regions as a single entity with a single host name.

The server instances may coexist on the same LPAR, but it is common practice to have a server instance per LPAR, thus utilizing the high availability functions of the sysplex to provide the optimum service. Figure 1-3 shows the concept.

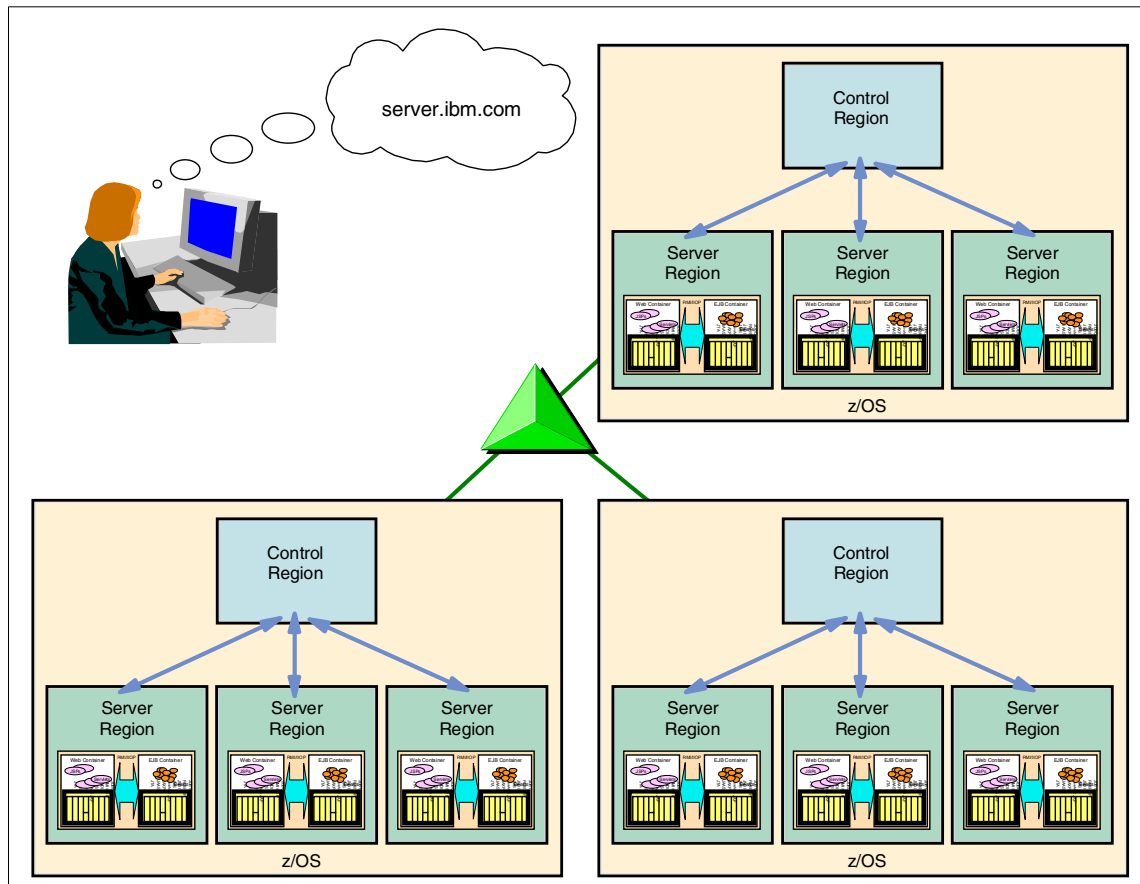


Figure 1-3 WebSphere server and server instances

A further refinement is to separate different e-business applications across different servers. Now there are multiple servers in the sysplex, comprised of multiple server instances running on each LPAR. The whole collection of servers is called a *node*. See Figure 1-4 on page 14.

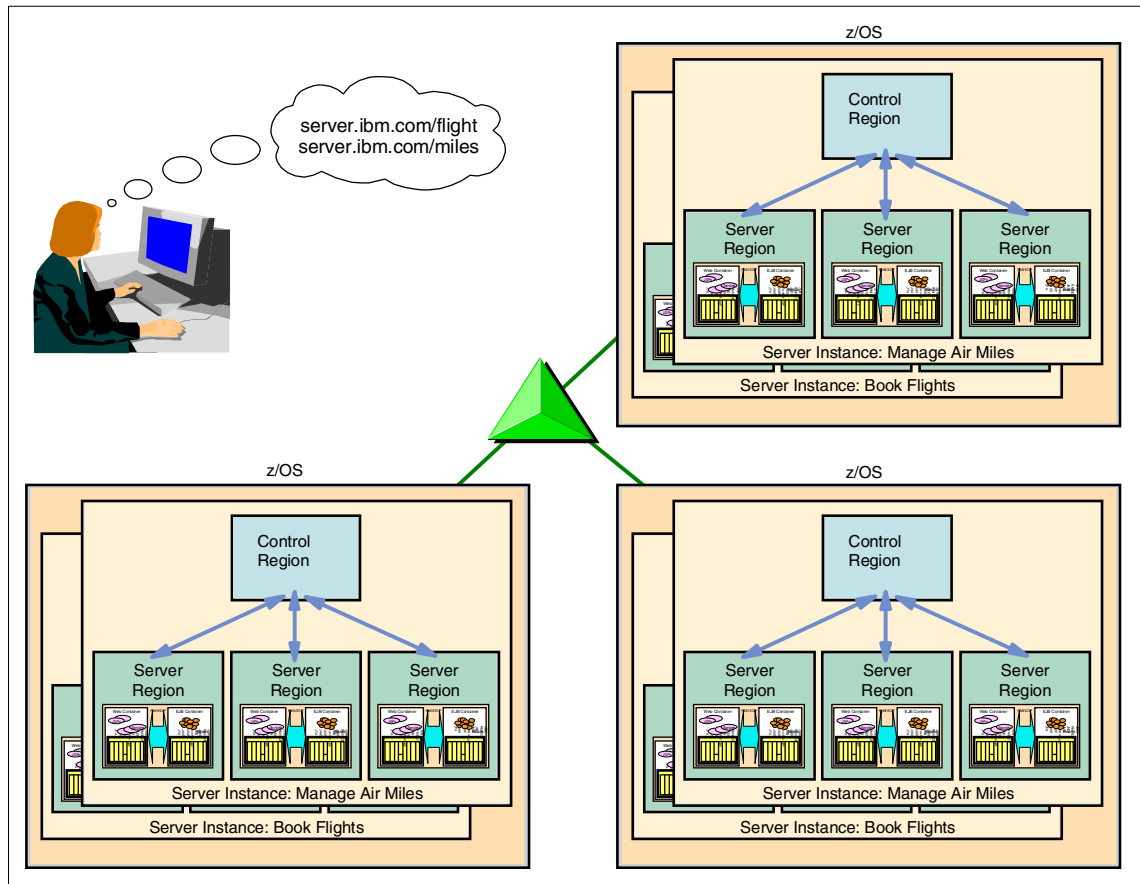


Figure 1-4 Multiple WebSphere servers in a sysplex node

The issues we now face are:

- ▶ How to present the same image (host name) to the clients from each server
- ▶ How to distinguish between different servers in the sysplex
- ▶ How to select the correct server instance if there is a session affinity between the client and the server—for example, if consecutive HTTP connections are part of the same transaction and must be handled by the same instance

Now we need a sophisticated connection distribution mechanism that can take account of these things, as well as input from WLM. The recommended method is described in 1.5, “Putting it together: a typical customer installation” on page 16.

1.4 System administration model

With such an application server environment, which is potentially very complex, the need arises for certain support services to assist both applications and system administrators to accomplish their tasks. These support services run in another set of address spaces, namely:

- ▶ The Management Server acts as the interface between the administrative client and the WebSphere environment. The administrative client is known as the Systems Management End User Interface (SMEUI). Like the Application server, the Management Server consists of a control region and some server regions. Note that the SMEUI client for z/OS is very different from the administrative client for the distributed WebSphere platforms.
- ▶ The LDAP server acts as a directory for EJBs and servlets.
- ▶ The Naming server, also comprising a control region and some server regions, is used to locate J2EE objects.
- ▶ The Interface Repository (IR) servers have been superseded in function by the Naming servers. However, they are always started automatically by z/OS WebSphere.
- ▶ The WebSphere Daemon starts the Management, Naming, and IR servers, and helps to locate services delivered by those servers.

Figure 1-5 on page 16 illustrates the setup of a typical set of support servers in a high-availability sysplex. Each LPAR always contains both a set of application servers and a set of support servers, but only the support servers are shown for clarity.

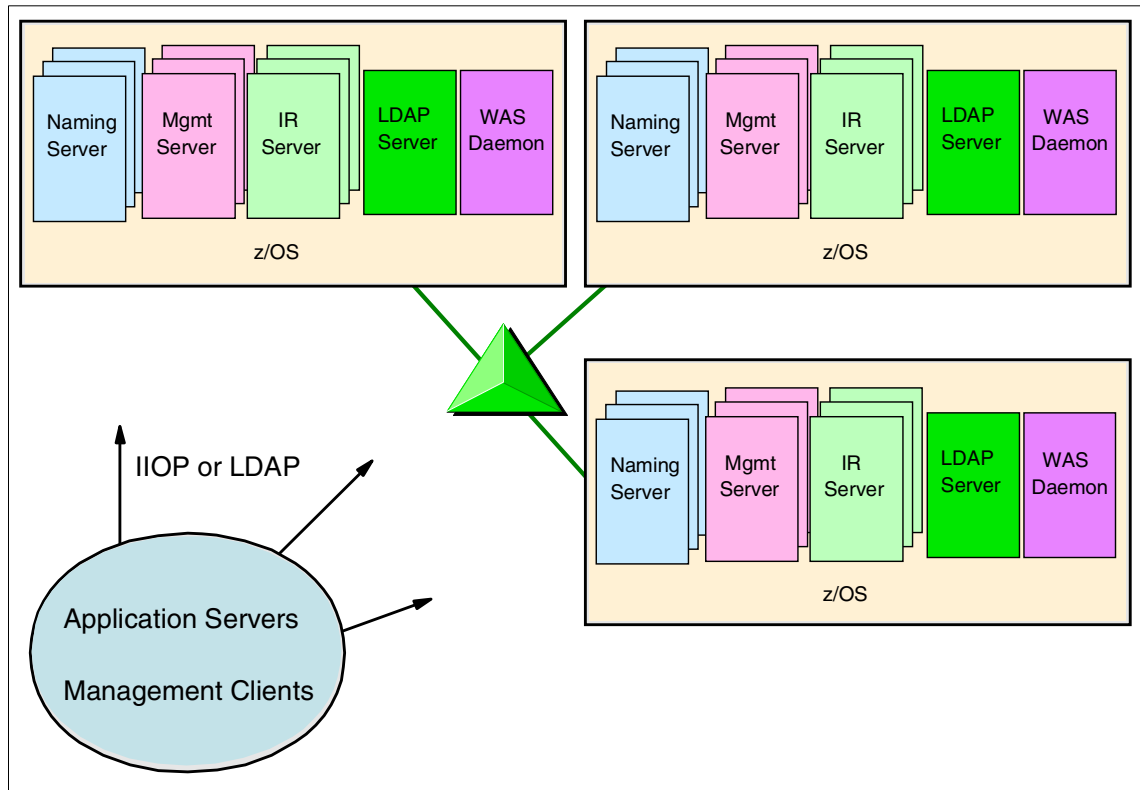


Figure 1-5 WebSphere support servers

All the above servers (except LDAP) communicate with each other using IIOP over TCP connections. They, too, can be configured for high availability just as the application servers can. Since their performance is not as interesting as the performance of the application servers, we concentrate on the application servers for the remainder of this book.

1.5 Putting it together: a typical customer installation

A WebSphere installation on z/OS can be as clever and as sophisticated as you like, but it still needs clients, and client access, in order to fulfill its purpose. In other words, it needs a TCP/IP network and some form of intelligent load distribution mechanism. The diagram shown in Figure 1-6 on page 17 depicts a setup that combines high availability with WLM-assisted workload balancing.

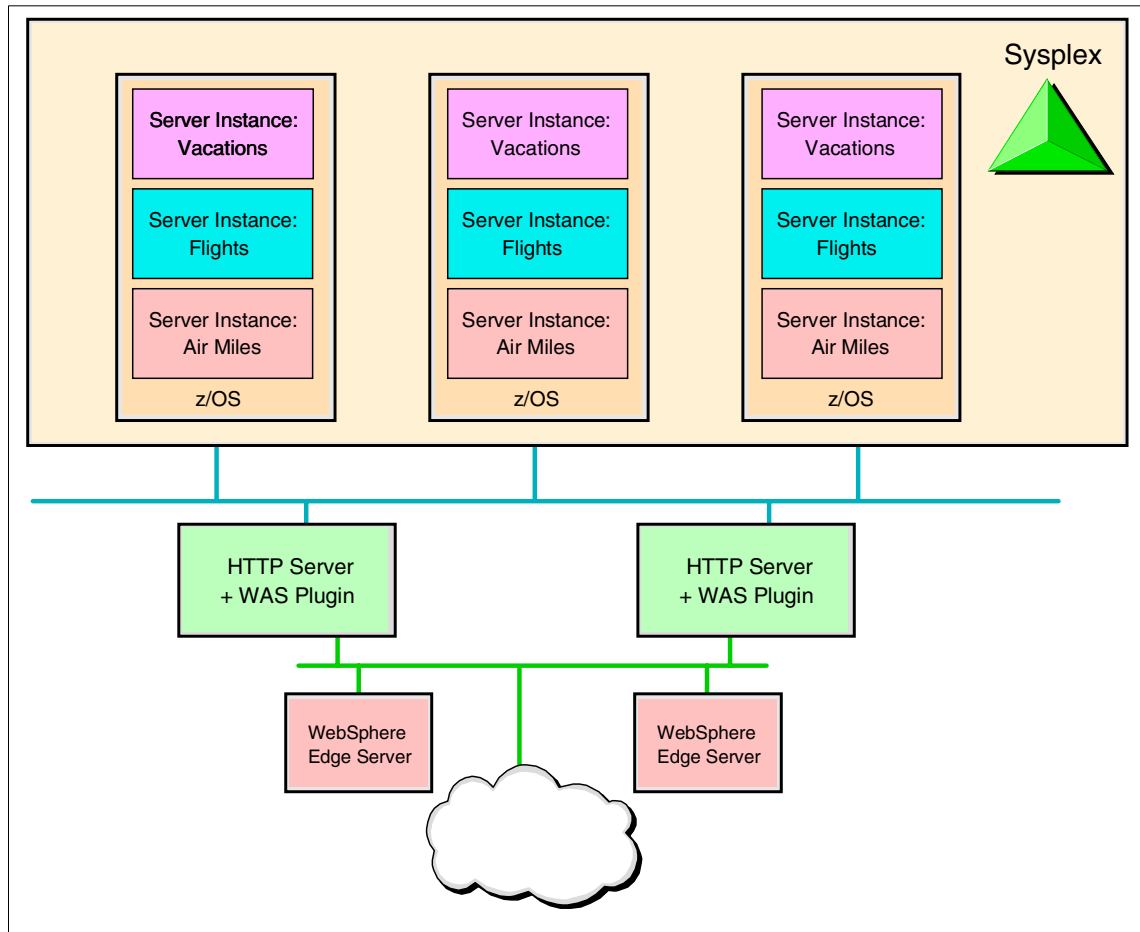


Figure 1-6 Typical WebSphere installation on z/OS

Here there are several distinct servers running on the z/OS sysplex, all listening for HTTP connections. The requirements are:

- ▶ To distribute incoming connections between the servers, based on WLM advice
- ▶ To check for session affinity, and to override WLM-based distribution if an affinity exists between a client and a server instance
- ▶ To distinguish between separate servers based on the client's input—usually a particular URI.
- ▶ To ensure high availability

This is accomplished by a combination of the following:

- ▶ Sysplex Distributor on z/OS, which balances incoming connections among available server instances based on WLM input. However, it distinguishes between server instances by port number, not by URI. Also, it does not provide any affinity between client and server; it cannot recognize when a client needs to access the same server on consecutive HTTP connections.
- ▶ The IBM HTTP servers with the WebSphere plug-in on the outboard servers, which can check the incoming URI and translate it to a port number. They can also check cookies to determine if session affinity exists, and if so, bypass Sysplex Distributor, forwarding requests directly to specific server instances.
- ▶ The IBM WebSphere Edge Servers (one primary and one backup), which distribute incoming connections between the HTTP server instances. They can also act as caching proxies, relieving the application servers of the tedious task of serving static pages.

Thus, a client connecting to the IP address of the WebSphere Edge Servers is assured of service from the correct z/OS server, and of continuing service if any component on the path fails.

In an environment where Internet access is available to the WebSphere sysplex, there would also be firewalls in the picture; these have been omitted for simplicity.

Refer to the redbook *Enabling High Availability e-Business on zSeries*, SG24-6850, for an in-depth discussion of this configuration.

1.6 Performance components

From the above discussion, you will have concluded that WebSphere Application Server on z/OS is not the simplest environment in which to perform performance investigations. There are very many factors that could adversely affect response times. Some of them can be easily identified by the tools described in this book, and some of them cannot. In this section we inspect the route taken by a typical Web transaction to identify the potential performance bottlenecks.

For a detailed and up-to-date list of tuning recommendations, see the latest edition of *WebSphere Application Server V4.0.1 for z/OS and OS/390: Operations and Administration*, SA22-7835. The major points in this section are a brief summary from Chapter 9 of SA22-7835-05.

This book is frequently updated; the latest version may be found at:

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

1.6.1 The TCP/IP network

The network is the first thing a user's request sees when it leaves the browser. It is also the least responsive to any tuning done by the installation, since much of it is outside your control. There are two things you can do to help:

- ▶ Good design of the environment immediately outside the sysplex: fast routers, fast switches, fast adapters (OSA Express), and efficient routing.
- ▶ Optimization of the z/OS TCP/IP stack: ensure that TCP buffer sizes are large enough, that MTU sizes are as large as possible, and that enough sockets are available for all the connections that need to be handled.

1.6.2 zSeries server

The obvious consideration is that WebSphere applications need hardware resources, memory, and CPU power. Less obvious factors are:

- ▶ Java applications use IEEE floating point instructions extensively. Prior to the 9672 G5 servers (predecessors to the zSeries), these instructions were emulated and performance could be adversely affected.
- ▶ If your Web site uses SSL encryption, this too is a heavy user of processing power. Hardware features available on the zSeries servers give them an advantage.

1.6.3 z/OS

As soon as the user's request hits the zSeries hardware, everything from that point onwards is under the control of z/OS until the response is sent back.

Recommendations include:

- ▶ Turn off all tracing unless absolutely necessary.
- ▶ Turn off recording of systems management facility (SMF) records other than the ones you need. Some of the WebSphere performance tools make use of SMF, so a measure of SMF recording may be required.
- ▶ Put frequently used Language Environment® modules into the link pack area (LPA).

UNIX System Services

WebSphere is a UNIX System Services (USS) application, meaning that it runs in the UNIX environment under z/OS. In configuring USS, you tell it how many processes (address spaces), threads (tasks), sockets, and users it is expected to handle. WebSphere uses large numbers of these things.

USS also uses a hierarchical file system (HFS) similar to that implemented on UNIX and PC platforms. You should ensure that search paths for required files are optimized. Also, if you are sharing HFS files between LPARs, make as many files as possible read-only. Writing to shared files incurs a significant overhead. Also, make sure HFS files are mounted locally when possible.

WorkLoad Manager

WorkLoad Manager (WLM) is responsible for delivering the correct service level to each application and to each user, as determined by the installation. “Correct” is defined by the goals that you configure in WLM. Goals are generally of two kinds:

- ▶ Response time
- ▶ Velocity, meaning percentage of requested processor time allowed

The trick with WLM is to map a given piece of work to a defined goal. Recommendations include:

- ▶ Classify all regions except the WebSphere server regions as high velocity.
- ▶ The server regions should be given a reasonable velocity for starting up and other work, like garbage collection. The real application work is handled under the application environment. Classify this with a suitable response time with a percentile goal.
- ▶ In the WLM definition, do not limit the number of server address spaces that can be started for a subsystem instance in an LPAR. Specify `No Limit` in the definition for your application environment. You can place suitable limits on the number via WebSphere itself.

RRS

Registration services, context services, and resource recovery services (RRS) are three separate z/OS components, but it is sometimes useful to think of them as a single function called recoverable resource management services (RRMS), the z/OS syncpoint manager.

WebSphere for z/OS requires the use of the RRS Attach Facility (RRSAF) of DB2, which in turn requires that resource recovery services (RRS) be set up. RRS provides services to authorized resource managers, such as database programs and communications managers that manage distributed transactional communications.

The Recovery and Restart Services component uses a two-phase commit to log transactions so that data can be recovered after a failure. The main recommendation is:

- ▶ Write RRS log records to the Coupling Facility whenever possible. Ensure that enough storage is available for the RRS structure.

For more information on system level resource recovery in z/OS, refer to *z/OS V1R2.0 MVS Programming: Resource Recovery*, SA22-7616.

Security

Security in a WebSphere environment is administered by the Resource Access Control Facility (RACF®) or one of its equivalent products. Security is costly in resources, so the principle to adopt is to define only what is necessary.

Enable only those classes (RACF authorization groupings) that you need. In particular, if you are not using EJB security roles (which define the users that can invoke individual methods), disable the appropriate facility class.

LDAP

The z/OS LDAP server is used by WebSphere, and runs multiple threads (subtasks) to service requests. It must be configured with enough threads to support the workload (one per server region is recommended).

JVM

The Java Virtual Machine interprets (or, as recommended for best performance, compiles Just In Time) the Java classes. Java spreads its work around in its storage (the Java Heap) and periodically cleans it up (garbage collection). Too small a heap size will lead to frequent garbage collection and poor performance. Recommendations include:

- ▶ Monitor the garbage collection cycles and define a sufficient heap size.
- ▶ Run with the JIT compiler active.
- ▶ Set CLASSPATH (the Java equivalent of a list of concatenated libraries) to point to the most frequently used classes first, and to omit classes not used.
- ▶ Keep up to date with PTFs, since many of them have performance enhancements.

WebSphere

The structure of the WebSphere server environment itself has many options that allow you to optimize performance within the available hardware resources. For example:

- ▶ How do you split the applications between servers?
- ▶ How many LPARs/instances for each server?
- ▶ How many server regions allowed per instance?

- ▶ Do you let a server region run multiple threads, or process one transaction at a time within each server region?

Some general recommendations for WebSphere are:

- ▶ Put as much of the code as possible into the LPA, and the rest in the link list. This will eliminate unnecessary searching of libraries.
- ▶ Make sure that enough storage (both real and virtual) is available. WebSphere is a heavy user of storage.

Containers run and manage the EJBs, JSPs, and servlets. Many of the properties associated with the containers can be tuned to improve efficiency. In particular, the behavior of pools of resources can be adjusted in terms of when pool elements get reused.

Connectors

Similarly, each type of connector has its own unique tuning requirements. Getting them right can prevent unnecessary data movement, unnecessary translation, idle resources being unavailable and so on.

Subsystems

The furthest point reached by a Web transaction from the client is usually the application that supplies the business data via the connector. Often the application was written long before the advent of J2EE and requires a connector to make it play in the Web environment. Sometimes you get the up-to-date solution, for example a DB2 database call made via JDBC. The most common z/OS applications to be found include:

- ▶ DB2 is a relational database that spans many platforms and integrates data on all of them. In terms of address spaces, threads, communication options and general complexity, it is almost the equal of WebSphere. Tuning DB2 for optimum data retrieval is a whole redbook in itself. One recommendation that is particularly important for WebSphere is to define sufficient DB2 threads; WebSphere uses a lot of them. It is important to note that, even if WebSphere applications do not use DB2, the WebSphere configuration data is in a DB2 database.
- ▶ Message Queueing (MQ) is a subsystem that manages the transmission of messages from place to place. Although the concept sounds simple, the reality is not; you will find MQ running in several address spaces and communicating with various distributed platforms. Tuning the storing and forwarding of messages is not a trivial task, although probably easier than tuning a relational database.
- ▶ Customer Information Control System (CICS) is a popular, long-standing transaction processing system. Simpler in structure than WebSphere or DB2,

it nevertheless can occupy a large number of address spaces spread across a sysplex. As with many subsystems, one of the major tuning options in CICS is to optimize the method by which transactions are logged.

- ▶ Information Management System (IMS) is a more complex, but more robust, transaction management system than CICS. It also comes with its own database system, DL/1 (Data Language One).

1.6.4 The application

Last, but not least, comes the actual coding of the Web application. To quote from the Operations and Administration Guide: “Badly designed or written application code makes the largest contribution to poor overall performance”.

Traditional z/OS applications were developed by highly paid, highly skilled specialists, and have been finely tuned over many years. Most of them make efficient use of system resources and after years of fine-tuning are less likely to be responsible for performance problems.

With the advent of e-business and Web-based applications, many new Java applications have been developed in the fastest possible time in order to present them to the world before competitors can respond. But the cost saved in development time and effort is reflected elsewhere: expensive consultants to review and tune the applications, and/or extra hardware to compensate for the inefficient code.

Nothing comes for free; the installation has to decide where to spend its money: development, tuning/fixing, or extra hardware resources. This book cannot tell you which option to take, but it may help you determine whether one of them is necessary.



WebSphere and z/OS, walking the performance path

This chapter provides an introduction to performance of WebSphere on z/OS. It describes the data that can be used to assess the performance and presents a general methodology for identifying the source of performance problems.

2.1 Introduction to performance and terminology

Performance

The performance of a server can be defined as a measure of how well it carries out a task. For a computer system or application we usually take this to mean how fast it carries out the task, but it can also include a measure of how many tasks it can complete in a given time.

Response time

According to the *IBM Dictionary of Computing*, which cites *International Organization for Standardization Information Technology Vocabulary* as the source:

“The elapsed time between the end of an inquiry or demand on a computer system and the beginning of a response; for example, the length of the time between an indication of the end of an inquiry and the display of the first character of the response at a user terminal.”

A Web user's view may be different from this. Consider the case where a servlet or JSP returns an HTML page that includes many GIFs, etc. The user is likely to view response time as the time between clicking their mouse until the resulting page is completely rendered in the browser. The previous definition would stop the response time clock when the first byte of HTML is received by the browser, not when the page has been completely rendered.

If you use an HTTP Server external to WebSphere to serve your static content, it may be impossible (or, at least, very difficult) to find a means of measuring response time in the same manner that a Web user perceives it. You would need to be able to measure the combined time to serve all the elements of the generated HTML page. If you use WebSphere to serve your static content, the servlet/JSP request and each GIF etc. referenced in the HTML would appear as separate items that have to be combined in some manner to form the complete response time. Luckily, this is rarely a problem because there are a number of points in the network and browser where static content will be cached.

RMF™ reports for WebSphere on z/OS measure the time between work being queued by a Control Region and that piece of work being completed in the Server Region. We will use this definition of response time in this chapter.

Throughput

This is a measure of the amount of work going through a system in a given time. Typically this may be measured as the number of transactions per second. As with response time, our measurement of throughput with WebSphere on z/OS is measured based on the work completed by the Server Region.

Transaction

A strict definition of transaction is “logical unit of work”. When one transfers money from one account to another, it is to be removed from the first account and then added to the second. The transaction includes both these processes, and they must both complete successfully for the transaction to be considered complete. A mechanism is also required to ensure that if one of the processes fails, the other is either not attempted or is also undone.

A customer at a browser may consider the complete process of selecting a book, entering payment and delivery details and then finalizing the purchase as a single transaction.

WebSphere considers each incoming request as a transaction. Each of the requests to WebSphere generated by the customer as they go through the process of buying a book will be treated as a separate transaction. Our discussion in this chapter uses the term transaction as viewed by z/OS Workload Manager and reported by RMF.

Hit rate

Often used as a measure of activity on a Web site. A hit is the retrieval of any single item from a Web server. Hence a Web page with four graphic items will actually count as five hits: one for the html page and one for each of the graphic items. Hit rate is the number of hits in a given time. While this does measure all the interactions between user and browser, it tends to hide the more valuable measure of the number of pages being accessed.

Page view rate

A more valuable measure than hit rate. This counts complete pages retrieved in a given time rather than all the individual elements.

Important: The above definitions should be understood when interpreting WebSphere transaction rate from an RMF workload activity report. For WebSphere on z/OS, RMF views each request as a transaction, whether it is a call to a J2EE application or a request to a static page element.

If WebSphere on z/OS is serving static pages, the transaction rate reported by RMF will in fact be closer to a measure of the hit rate.

If static content is served from another source, for example a WebSphere Edge server front end, and requests issued to the back-end application server are mostly for J2EE applications, then the value reported by RMF will be closer to the resulting Page View rate.

Number of clients and think time

The number of clients is the number of users connected to the Web site. However, as opposed to legacy applications, there is no direct relation between the number of clients connected and the load on the Web server. This is due to the heterogeneous nature of Web applications. In a traditional CICS or IMS application, users tend to be logged on working almost continuously. In a Web application, for example when buying a book, there tends to be more browsing while users evaluate the information that has been returned. This is *think time*. While the users are thinking, they are still effectively connected to the site but they are not driving work in WebSphere (although there may still be a session object from their previous interaction). Thus, in an application that tends towards long think time, there may be a large number of concurrent users, also called clients, but a low transaction rate in WebSphere application server.

Resource

This is any item that can be used in the execution of a transaction. This can be a physical resource (for example, CPU, memory) or a logical resource (for example, JDBC connection, a queue in WLM, etc.). When a WebSphere transaction accesses data in DB2 or CICS, it may also be convenient to refer to DB2 or CICS as a resource.

For a transaction to complete, it must be able to access all the resources it requires. For a transaction to perform well, there need to be enough of these resources available and they need to be available quickly enough. How much is enough? How quick is quick? There is no hard answer. It depends on your business requirements.

2.1.1 Setting your performance expectations

How many transactions per second should you expect from a given WebSphere on z/OS implementation? As any application, WebSphere applications are using system resources. One has to rely on a number of sources, some of which are unreliable and unrealistic, when setting your expectations.

These include:

- ▶ Monitoring and extrapolation

If you already have a running application, by taking appropriate measurements on a regular basis you will understand how your application performs in normal operation. Any deviation from this base line may represent a performance problem. Be careful when projecting forward from monitored data. The numbers may not scale in a linear manner, especially if you are already close to some limit. You need to carefully review logical resources as well as physical ones (e.g., CPU).

- ▶ Experience

Be careful when generalizing. Different applications may behave in very different ways. Benchmarks generally only tell you how well the benchmark environment was prepared, it will not guarantee how your application will behave in your real-life environment.
- ▶ Unrelated experience on other environments, systems or subsystems

WebSphere is not CICS. As explained earlier, they tend to serve different types of users, and you should not expect similar behavior. Although WebSphere Application server on z/OS is J2EE-compliant, applications will not necessarily behave in the same way as on WebSphere on distributed platforms.
- ▶ Business requirements

Unless backed by data which confirms that these requirements can be met in your environment, in reality this may be little more than a statement of intent.
- ▶ Load testing

Using workload simulation tools, such as WebSphere Studio Workload Simulator, you can evaluate how an application will behave in your environment as long as you can recreate a testing environment that matches the projected production environment.
- ▶ Capacity planning

Although there is very little information published on this topic, your IBM representative or authorized Business Partner has access to Technical Support to do pre-sales sizing estimates of your WebSphere Application on zSeries servers.

2.1.2 Performance management

There are many situations in performance management. Typically, they are

- ▶ Performance monitoring - seeing that everything is running smoothly
- ▶ Performance analysis - getting to the seat of problems
- ▶ System tuning - ensuring the best usage of resources
- ▶ Capacity planning - ensuring that you have enough resources

In this document, we address the first two points in a WebSphere environment.

What is a performance problem?

There are many views on what constitutes a performance problem. Most of them revolve around unacceptably slow response times or high resource usage, which we can collectively refer to as pain. The need for performance investigation and

analysis is detected by system indicators or users complaining about slow response.

Ultimately, you will have to decide for yourself whether a given situation is a problem worth pursuing or not. This decision will be based on your own experience, knowledge of your system, and sometimes politics. We will simply assume for the following discussions that you are trying to relieve some sort of numerically quantifiable pain in your system.

Generally, a performance problem is the result of some workload not getting the resources it needs to complete in time. Or, less commonly, the resource is obtained but is not fast enough to provide the desired response time.

A common cause of performance problems is having several address spaces, or threads (or tasks) compete for the same resource. These could be a hardware resource or a serially usable software resource.

For this document we assume that there is a potential performance problem when for an application in the WebSphere Server Region, response time measured in milliseconds per transaction, or throughput measured in transactions completed per second, does not meet your expectations.

Most definitions revolve around unacceptably high response times or resource usage. However, the definition of “unacceptably high” will vary from one installation to another. On z/OS, the peaks and troughs of other workloads on the same system image will impact WebSphere and vice versa.

The business may need to prioritize other workloads on the image at some point in time (for example, year-end batch processing), even though this will be detrimental to the performance of WebSphere applications.

The tools and techniques in this chapter will help you to identify where your resources are being consumed and why and where your application is experiencing delays.

They will not be able to tell you whether or not such answers are applicable to your situation. Ultimately this is a business decision.

2.1.3 How to know that you have a performance problem

Some indications of slow performance are:

- ▶ Complaints from users
- ▶ Service level objectives not being met
- ▶ Alerts from monitoring tools

- ▶ Unexpected changes in reported usage
- ▶ System resource indicators (for example, paging rates, DASD response)
- ▶ Expected throughput on the system not being attained

Most of these indications assume that some degree of monitoring is in place. Without monitoring, it is impossible to make objective judgements when comparing current performance to past performance or knowing what is normal for a given application. It is also impossible to objectively verify a user's complaint without knowing what is really happening on your system.

2.1.4 What to do about a performance problem

How and if you apply a solution is ultimately dependent on business priorities. If a proposed solution to a WebSphere performance problem requires taking resources from another application, a business decision will have to be made to determine whether or not this is a price worth paying. If the recommended solution involves extensive application recoding, the cost may not be justifiable if the application has a short life expectancy.

When considering cost effectiveness, consider the cost of users abandoning your site because it is too slow. They will take their purchases elsewhere and may never try your site again. You may never be aware of the business you are losing.

Make sure your performance expectations are realistic

Try to understand what can actually be achieved with your application given your hardware and software configuration. As discussed in 2.1.1, "Setting your performance expectations" on page 28, this is actually a difficult question to answer. The end point of the problem determination process may be to reset your expectations.

Don't cause it

Make sure that you follow published performance configuration guidelines.

We recommend that you run on a G5 server or later to avoid Java performance issues with IEEE floating point. Prior to G5, the IEEE floating point instruction set had to be emulated in software. From G5 and later, the instruction set is implemented in hardware. SDK1.3, which is required for WebSphere Application Server V4.0.1, makes more use of this instruction set than previous versions, so the impact will be more noticeable than before.

We recommend that you run with at least 512 MB of real storage. For more complex applications, you may need 1 GB or more of real storage. The definition

of “more complex” is rather vague, so it may be best to plan for 2 GB from the outset.

WebSphere 4.0.1 tuning recommendations can be found in chapters 9 - 10 of *WebSphere Application Server V4.0.1 for z/OS and OS/390: Operations and Administration*, SA22-7835. This manual is often updated, so you should check for availability of the latest version at:

http://www.ibm.com/software/webservers/appsrv/zos_os390/library.html

Check the IBM HTTP server recommendations. If you are performance conscious, you are probably using the HTTP Transport Handler running in the WebSphere control region. However, if you are also using the IBM HTTP Server, you should also check installation recommendations at:

<http://www.ibm.com/software/webservers/httpservers/doc/v51/2tabcont.htm>

Check for WebSphere performance information in APARs. The latest information can be found at:

<http://www.ibm.com/support/docview.wss?rs=404&conext=SS6LRK&uid=swg21063537>

e-business is a fast-moving world. Consequently there is a seemingly never-ending supply of maintenance. Keeping current with maintenance is more important than for more traditional workloads and will often bring improved performance.

A significant update to the WebSphere product code, maintenance level W401400, was made available in September 2002. As well as a general maintenance roll-up, this introduced a number of new features that aid performance. We highly recommend installing this maintenance level.

Fix it, if you can

Generally, a performance problem is the result of some workload not getting the physical or logical resources it needs to complete in a timely manner, so the solution is to make more resources available.

If the solution involves making more hardware resource available to the application, you can do this by:

- ▶ **Buying more**

If there is no other means of making your application performance meet your expectations, add more resources. At times this may also be more cost effective than recoding a badly-written application. Beware of induced costs.

- ▶ **Stealing it**

Take it from a less important application. Here the price to pay is a lower service to the application from which the resources are stolen.

- ▶ Using less, for example:
 - Fix badly written code.
 - Revise poorly performing SQL queries or add appropriate indexes to improve fetches from databases.

The cost here is development people. While the cost may be higher, it may be less visible than buying new hardware.

Live with it

If none of the above options are technically or financially possible, it will be necessary to change your expectations. At least you will know why the performance is not meeting your previous expectations. Your users may be disappointed by the answer, but at least you will be able to give them facts and convince them that the situation is understood and under control. Changing the perception may be an important factor in user satisfaction.

2.2 Workload Manager controls

WebSphere Application Server V.4.0.1 for z/OS uses the Workload Manager (WLM) to manage the number and performance of application server regions in z/OS (see Figure 2-1 on page 34):

- ▶ For each server defined to WebSphere, an Application Environment must be defined in the WLM panels, which 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.

See Chapter 9 in *WebSphere Application Server V4.0.1 for z/OS and OS/390: Operations and Administration*, SA22-7835, for more details.

Managing the number of application server regions

Each WebSphere application server can have one or multiple server regions per server instance based upon the settings defined in the WLM application environment.

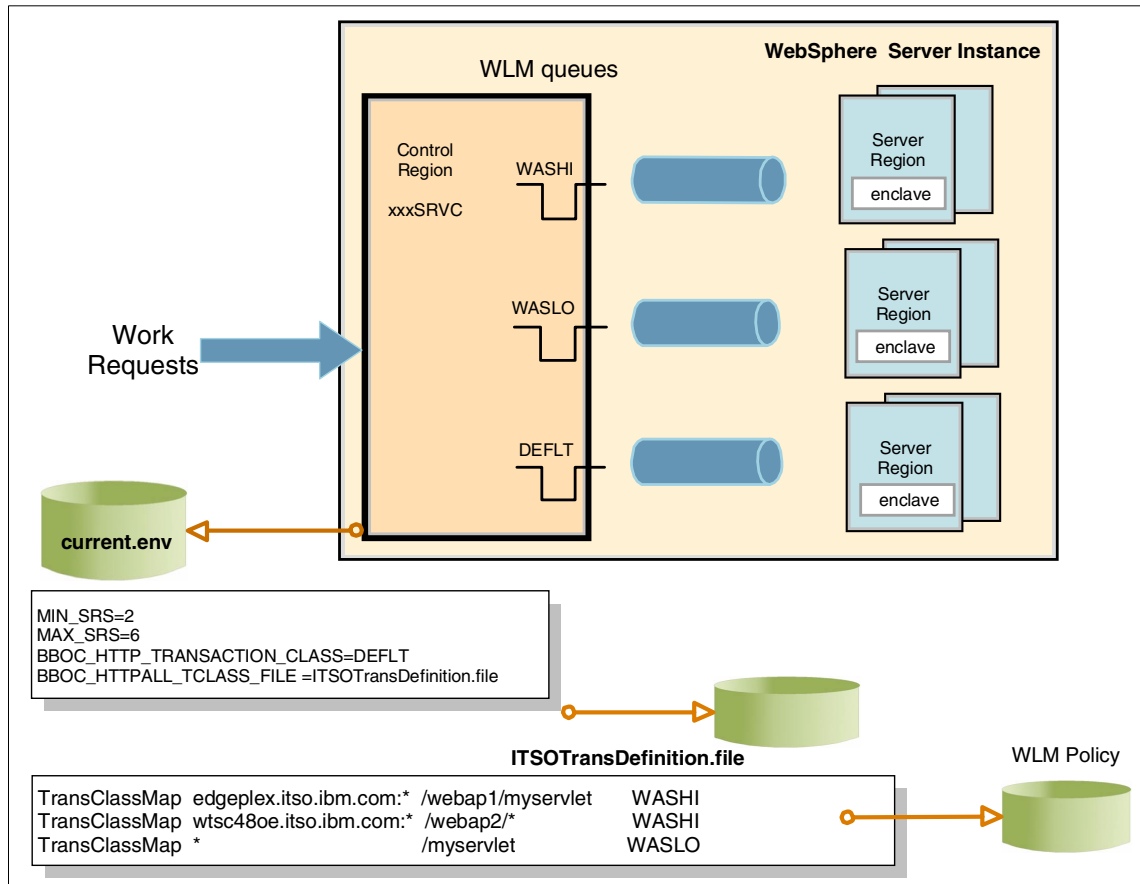


Figure 2-1 WebSphere runtime and Workload Manager

How many server regions are created depends on WLM determination of how the work is meeting its performance goals, the importance of the work compared to other work in the system, the availability of system resources needed to satisfy those objectives, and a determination by 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 maximum. You can override the maximum and minimum number of server regions that WLM will start with two parameters in the `current.env` file managed by the SMEUI. For each server, you can specify `MIN_SRS` and `MAX_SRS` to set boundaries on how many server regions WLM will start.

- ▶ `MIN_SRS` is used to start up a basic number of server regions before the day's work arrives. This can save time in waiting for WLM to determine that more server regions are needed.

- ▶ MAX_SRS is useful to cap the number of address spaces started by WLM if you determine that excessive server regions could contribute to service degradation.

Beyond this APPLENV use, MIN_SRS and MAX_SRS also have an influence on the dispatching of transactions managed by WLM.

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 only serves one service class at a given time. To ensure that you do not limit the parallelism of execution under full load, MAX_SRS should be set at least as large as the number of service classes defined.

If you specify MAX_SRS too low, there will be less servers available than WLM queues. The result may be a queue bottleneck under full load conditions, since workload manager 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.

For a detailed discussion on WLM and MAX_SRS, see document number TD100887 on the Technical Sales Library Web site at:

<http://www-1.ibm.com/support/techdocs/atmastr.nsf/WebIndex/TD100887>

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 according to the CB service classification rules.

The classification can be based on the following classification criteria (see Figure 2-2 on page 36):

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

```

Subsystem-Type  Xref  Notes®  Options  Help
-----
Command ==> _____ Modify Rules for the Subsystem Type _____ Row 1 to 10 of 10
SCROLL ==> PAGE

Subsystem Type . : CB          Fold qualifier names?  Y  (Y or N)
Description . . . WebSphere App Server

Action codes:  A=After      C=Copy      M=Move      I=Insert rule
               B=Before    D=Delete row R=Repeat   IS=Insert Sub-rule
                                           More ==>

Action      -----Qualifier-----          -----Class-----
Type      Name      Start      Service      Report
-----
_____ 1 CN      FMISRV*    _____  _____  WASE
_____ 1 CN      FMESRV*    _____  _____  WASE
_____ 1 CN      OMESRV*    _____  _____  WASE
_____ 1 CN      OMTSRV*    _____  _____  WASE
_____ 1 CN      INTSRV*    _____  _____  WASE
_____ 1 CN      INESRV*    _____  _____  WASE
_____ 1 TN      WASLO     _____  _____  WASE
_____ 1 TN      WASDF     _____  _____  WASE
_____ 1 TN      WASHI     _____  _____  WASE
*****
F1=Help  F2=Split  F3=Exit  F4=Return  F7=Up  F8=Down  F9=Swap
F10=Left F11=Right F12=Cancel

```

Figure 2-2 WLM definitions of the server regions, CB subsystem

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

You can further use the virtual host name, port number, or URI template to map the HTTP request to a transaction class with a filtering file specified in the `BBOC_HTTPALL_TCLASS_FILE` variable. Here is an example:

```

TransClassMap haplex1.itso.ibm.com:*      /webap1/myservlet  WASDF
TransClassMap haplex1.itso.ibm.com:7080  *                  WASHI
TransClassMap *:7070*                      /trade/*          WASDF
TransClassMap *                            /eITS0/*          WASDF

```

```

                                Modify a Service Class                                Row 1 to 2 of 2
Command ==> _____

Service Class Name . . . . . : WASHI
Description . . . . . . . . . : LSA510 WAS 200MS RT
Workload Name . . . . . . . . : WAS          (name or ?)
Base Resource Group . . . . . : _____ (name or ?)
Cpu Critical . . . . . . . . . : NO          (YES or NO)

Specify BASE GOAL information. Action Codes: I=Insert new period,
E=Edit period, D=Delete period.

      ---Period---  -----Goal-----
Action # Duration  Imp. Description
-----
   1          1    90% complete within 00:00:00.200
***** Bottom of data *****

F1=Help    F2=Split    F3=Exit    F4=Return  F7=Up  F8=Down
F9=Swap    F10=Menu Bar F12=Cancel

```

Figure 2-3 WLM definition, CB Service Class

It is recommended that you define WebSphere transaction service classes using a percentage response time objective, as illustrated in Figure 2-3.

- ▶ It is the technical indication to WLM of your requirement. A response time objective is usually consistent with the business requirement of a Web application. The response time value may be adjusted depending on the type of application.
- ▶ This option automatically generates response time distribution information that is reported through an RMF report (see “Response time distribution” on page 52). You will find this option useful later on, when having to troubleshoot response time issues.

Server region address space classification

In addition, we recommend that you define a report class for the server address space activity. This will allow 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 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 WLM determines they are needed, but use an importance and velocity lower than the enclave classification (Figure 2-4). Again, we recommend to define a reporting class in order to isolate the activity into a specific workload report.

```

Subsystem-Type Xref Notes Options Help
-----
                Modify Rules for the Subsystem Type      Row 1 to 16 of 61
Command ==> _____ SCROLL ==> PAGE

Subsystem Type . : STC          Fold qualifier names?  Y (Y or N)
Description . . . Use Modify to enter YOUR rules

Action codes:  A=After      C=Copy      M=Move      I=Insert rule
                B=Before    D=Delete row R=Repeat  IS=Insert Sub-rule
                                   More ==>

                -----Qualifier-----                -----Class-----
Action  Type   Name   Start                Service   Report
-----  ---   ----   ----                -
          1   TN    HWS710*  ___                DEFAULTS: SYSSTC  OTHER
          1   TN    FMESRVS* ___                IMSTL           WASI
          1   TN    FMISRVS* ___                VEL80           WASS
          1   TN    INESRVS* ___                VEL80           WASS
          1   TN    WSESRVS* ___                VEL80           WASS
          1   TN    OMESRVS* ___                VEL80           WASS
          1   TN    HAO*     ___                _____       WAS
          1   TN    FMESRV*  ___                VEL85           WAS
          1   TN    INESRV*  ___                VEL85           WAS
          1   TN    OMESRV*  ___                VEL85           WAS
F1=Help  F2=Split F3=Exit  F4=Return F7=Up    F8=Down  F9=Swap
F10=Left F11=Right F12=Cancel

```

Figure 2-4 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, etc. Therefore, control regions should also be classified in SYSSTC or a high velocity goal.

2.3 Gathering WebSphere performance information

2.3.1 SMF records

Performance information

System Management Facility (SMF) is a good source of information for system and subsystem performance data. It can record information from most system components, including 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	USS HFS information
Record type 103	HTTP Server information
Record type 100-102	DB2 statistics, accounting, performance.
Record type 110	CICS TS Statistics
Record type 115, 116	WebSphere MQ Statistics.
Record type 118, 119	TCP/IP Statistics.
Record type 120	WebSphere Application Server information. The SMEUI allows to switch recording ON/OFF selectively for activity records or interval records.

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

2.3.2 RMF reports

This section summarizes a step-by-step approach to identify where to obtain useful information, using standard RMF reports and simple arithmetic. RMF reports do not give application information, but they can be used to obtain system and workload characteristics.

To simplify the monitoring, we logically grouped the WebSphere activity into predefined report classes:

- ▶ WAS for WebSphere infrastructure, control region, SM, naming servers
- ▶ WASS for server regions
- ▶ WASE for e-business workloads running in the server regions in enclaves
- ▶ WASC for CICS server called upon by WebSphere transactions
- ▶ WASD for DB2

- ▶ OTHER for other activity not directly related to our WebSphere environment. Since our exercise was to illustrate a production environment as opposed to a lab “controlled” environment, this was done to isolate started tasks, systems management tasks, TSO users, etc., that were concurrently active on the sysplex.

Although RMF provides a graphic user interface with RMF PM Java edition, we chose to illustrate our approach using the traditional RMF post processor. One reason is that the post processor is easier to document in a generalized way since it uses predefined formats, whereas RMF PM provides GUI-customized views that tend to vary with each installation.

There are two useful types of RMF reports:

- ▶ Summary and CPU reports, which give system-wide information
- ▶ Workload activity reports, which provide information on workloads

The first thing you probably want to do before you go deep into WebSphere application tuning is to quantify as precisely as possible where z/OS resources are currently used. If you identify an imbalance or a resource constraint at the system level, you probably want to correct it first. There is little chance that you can fix a WebSphere application problem if the system is not reasonably well tuned.

Most of the information on resource utilization can be quickly obtained from RMF reports.

CPU	The partition data report gives the logical partition view. The summary report and CPU report show the z/OS system view, while the workload report provides a breakdown by workload type.
Storage	The summary and CPU reports, show the z/OS system view. The workload report provides storage allocation information by workload type.
I/O Activity	The summary report, CPU report, and IOQ report shows system level indicators. The workload report provides information by workload type.

If a problem is suspected from these high-level reports, additional resource reports such as channel, IO activity, paging, virtual storage reports, can be further investigated.

Measurements

Two points should be considered. One is the influence of the measurement tool on the measured environment, the other is the choice of the measurement period.

Some of the data reported by RMF comes from event counters. But much of the data in the RMF paging, virtual storage, CPU or I/O queuing, reports is statistically sampled. Because, according to statistical theory, the accuracy of sampled data increases with the number of samples taken of random events, you would expect to observe more precise results with decreased CYCLE time (for a fixed INTERVAL value), or with increased INTERVAL length (for a fixed CYCLE value).

However, pure statistical predictions are not always applicable to a software measurement tool because the assumptions on which they are based (unbiased random independent samples and an infinite population) might not hold in an operating environment. Bias might occur because the tool samples internal indications of external system events.

The independence assumption becomes less and less realistic as CYCLE gets very small. As CYCLE gets smaller, each sample is more likely to find the system performing the same functions as in the previous sample; therefore, the new sample adds little additional information. The use of a smaller CYCLE value (while holding INTERVAL constant) should not be detrimental to accuracy, but any increase in accuracy might be of questionable benefit when compared with the system overhead that is introduced.

In our measurements, which were run on a 2064 (either model 2C7 or 1C8; the configuration changed during the project) we used a cycle time of one second and the measurement interval was set to 5 minutes. That translates into 300 samples per measurement interval.

Although it is not expected that one would keep a 5-minute interval in a real production environment, it is recommended not to set the measurement interval too high when running in troubleshooting mode. A good compromise may be 15 or 20 minutes (a 15-minute interval using a one-second cycle would lead to 900 measurement samples per interval). Although extending the interval beyond 30 minutes is possible, it will average the results in such a way that many short peaks may no longer be visible in the reports.

The second decision point relates to the choice of the measurement analysis period.

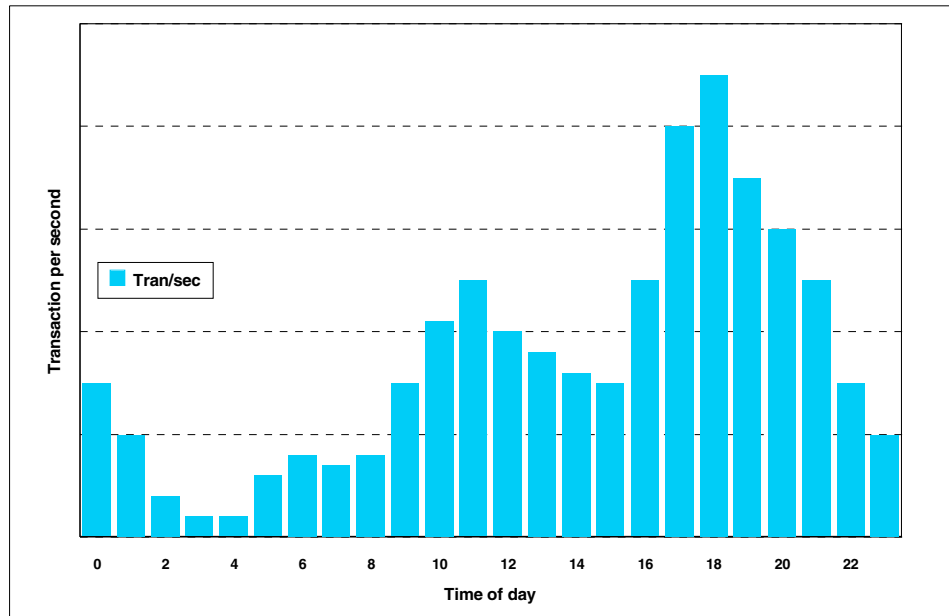


Figure 2-5 A typical daily activity profile

The graph in Figure 2-5 illustrates a daily activity profile, but the same reasoning would apply to a weekly or monthly profile. What one needs to determine is what is a representative period. Note that this not a technical-only decision; some knowledge of the business background is usually required to determine which part of the application activity cycle is meaningful to the performance analysis.

Running the RMF post processor

The JCL used to produce the RMF reports is shown in Example 2-1.

Example 2-1 JCL for running the post processor

```
//RMFRPT52 JOB (999,P0K), 'FRANCK', CLASS=A, REGION=4096K,
//          MSGCLASS=T, TIME=90, MSGLEVEL=(1,1), NOTIFY=&SYSUID
//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))
//SORTWK02 DD DISP=(NEW,DELETE),
//          DSN=&&WK2, UNIT=SYSALLDA, SPACE=(CYL, (50,50))
//SORTWK03 DD DISP=(NEW,DELETE), DSN=&&WK3,
```

```

//          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)
//* REPORTS (CHAN)
//* REPORTS (ENQ)
//* REPORTS (IOQ)
//* REPORTS (PAGING)
//* REPORTS (DEVICE(DASD))
//* REPORTS (OMVS,HFS)
//*
//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)
//* SYSRPTS (WLMGL(POLICY(FRANCK.LSM301_1)))
//*
//SYSIN    DD      *
      RTOD(0000,2400)
      STOD(0000,2400)
      SUMMARY (INT)
      SYSRPTS (WLMGL(RCLASS(WAS*,OTHER,SYS*)))
      SYSRPTS (WLMGL(POLICY,SCPER(WAS*)))
      SYSOUT(T)

```

Partition Data Report

The RMF Partition Data Report is imbedded as part of the CPU Activity Report when the server is running in LPAR mode. To access the report information for all partitions you need to be authorized; this is done from the zSeries server Hardware Management Console (HMC) by enabling the Global Performance Management Control setting in the partition activation profile.

The Partition Data Report section contains header information, partition data, logical partition processor data, and average processor utilization percentages.

PARTITION DATA REPORT									
z/OS V1R3		SYSTEM ID SC48		DATE 11/17/2002		INTERVAL 05.00.509			
		RPT VERSION V1R2 RMF		TIME 16.54.59					
MVS PARTITION NAME		A11							
IMAGE CAPACITY		171							
NUMBER OF CONFIGURED PARTITIONS		15							
NUMBER OF PHYSICAL PROCESSORS		13							
CP		7							
ICF		6							
WAIT COMPLETION		NO							
DISPATCH INTERVAL		DYNAMIC							
----- PARTITION DATA --				-- AVERAGE PROCESSOR UTILIZATION PERCENTAGES --					
-----MSU--				LOGICAL PROCESSORS		--- PHYSICAL PROCESSORS ---			
NAME	S	WGT	DEF	EFFECTIVE	TOTAL	LPAR	MGMT	EFFECTIVE	TOTAL
A1	A	180	45	4.65	5.05	0.11		1.33	1.44
A2	A	10	30	4.52	4.91	0.11		1.29	1.40
A3	A	180	0	4.52	4.92	0.11		1.29	1.41
A4	A	10	0	0.78	0.92	0.04		0.22	0.26
A5	A	10	45	4.12	4.52	0.11		1.18	1.29
A6	A	10	0	10.19	10.59	0.12		2.91	3.03
A7	A	10	45	4.52	4.97	0.13		1.29	1.42
A8	A	10	0	3.99	4.38	0.11		1.14	1.25
A9	A	10	0	3.60	3.99	0.11		1.03	1.14
A10	A	10	0	3.31	3.71	0.11		0.95	1.06
A11	A	180	0	89.93	90.16	0.07		25.69	25.76
A12	A	10	50	0.91	1.04	0.04		0.26	0.30
PHYSICAL								4.19	
TOTAL						5.36		38.58	
								43.95	

Figure 2-6 Partition Data Report (partial view)

From the example shown in Figure 2-6, we can quickly see that:

- ▶ The running partition is A11.
- ▶ It is using 90.16% of its logical CPs, which translates into 25.76% of the server CP capacity.
- ▶ The zSeries server CPs are only used for 43.95% of the time.

The Physical Management Time reported by RMF, in the *PHYSICAL* line, indicates the amount of processor time required by LPAR to manage all active logical partitions. The partition named PHYSICAL does not exist, the line is created by RMF for reporting purposes.

z/OS V1R3		SYSTEM ID SC48		DATE 11/17/2002		RPT VERSION V1R2 RMF		TIME 16.54.59	
MVS PARTITION NAME								A11	
IMAGE CAPACITY								171	
NUMBER OF CONFIGURED PARTITIONS								15	
NUMBER OF PHYSICAL PROCESSORS								13	
CP								7	
ICF								6	
WAIT COMPLETION								NO	
DISPATCH INTERVAL								DYNAMIC	
----- PARTITION DATA ----- -- LOGICAL PARTITION PROCESSOR									
----MSU---- -CAPPING-- PROCESSOR- ----DISPATCH TIME									
NAME	S	WGT	DEF	ACT	DEF	WLM%	NUM	TYPE	EFFECTIVE
A1	A	180	45	4	NO	0.0	2	CP	00.00.27.930
A2	A	10	30	4	NO	0.0	2	CP	00.00.27.137
A3	A	180	0	4	NO	0.0	2	CP	00.00.27.192
A4	A	10	30	1	NO	0.0	2	CP	00.00.04.705
A5	A	10	45	4	NO	0.0	2	CP	00.00.24.785
A6	A	10	0	9	NO	0.0	2	CP	00.01.01.222
A7	A	10	45	4	NO	0.0	2	CP	00.00.27.174
A8	A	10	0	4	NO	0.0	2	CP	00.00.23.989
A9	A	10	0	3	NO	0.0	2	CP	00.00.21.648
A10	A	10	0	3	NO	0.0	2	CP	00.00.19.890
A11	A	180	0	77	NO	0.0	2	CP	00.09.00.486
A12	A	10	50	1	NO	0.0	2	CP	00.00.05.489
PHYSICAL									

TOTAL									00.13.31.653
C1	A	DED	0	86		0.0	2	ICF	00.10.00.977
C2	A	DED	0	86		0.0	2	ICF	00.10.00.774
C3	A	DED	0	86		0.0	2	ICF	00.10.00.644
PHYSICAL									

Figure 2-7 Partition Data report and processing weights (partial view)

The logical partition Dispatch Time Effective indicated for each configured partition, as shown on Figure 2-7, is the sum of the z/OS captured time and the z/OS uncaptured time. The Partition LPAR Management Time is *not* a collected value, but is calculated by subtracting DISPATCH TIME DATA EFFECTIVE from the DISPATCH TIME DATA TOTAL.

Note that the flexibility brought by logical partitioning adds an additional level of complexity to the performance analysis: unless the logical partition is “capped”, the amount of CPU processing power that the partition can use can vary.

- The minimum CPU the logical partition is entitled to is determined by the processing weights set as part of the partitioning definition.

$$\text{Min LP CP share} = \text{Your LP weights} / \text{sum of all LP weights}$$

This will occur when other partitions require their full share of CP resource. In Figure 2-7 on page 45, which illustrates our test configuration, the sum of all WGT is 630, while our LP, A11 as indicated, has a processing weight of 180.

The guaranteed CP share is $180/630 = 28.57\%$ of the shared CPs.

- ▶ The maximum CPU the logical partition can use is fixed by ratio of the number of CPs defined in the partition to the total number of available CPs in the shared pool.

Max LP CP share = number of CPs / sum of shared CPs

This occurs when other partitions do not need their full share of CPU resource. Looking again at Figure 2-7 on page 45, there are 7 shared CPs available while our LP (A11) has 2 CPs defined.

The maximum usable CPU share is $2/7 = 28.57\%$ of the shared CPs.

In this example, we were able to align both the minimum and maximum values in order to simplify our tests, but in a real production environment this may not always be possible, nor desirable.

In addition, if the partition is part of an LPAR cluster, Workload Manager can dynamically adjust the number of logical processors and the weight of a logical partition. This allows the system to distribute the CPU resource in an LPAR cluster to partitions where the CPU demand is high (an LPAR cluster is defined as the set of logical partitions in a single server that belong to the same parallel sysplex).

Since the processing weights can be dynamically adjusted, either by operations personnel or by LPAR cluster management, remember to check their settings before you start a time-consuming workload analysis.

Note: All percentages indicated in the Partition Data Report are relative to the RMF time interval. As such, they accurately show the amount of time physical CPs were dispatched on behalf of LPAR or of a logical partition. However, these time-based figures do not take into account all processor costs of operating in LPAR mode and do not reflect the resulting processor power expressed in the Large System Performance Reference (LSPR) ITRs or MIPS.

The LPAR Capacity Estimator (LPARCE) tool should be run to estimate the impact of the logical partition configuration on the processing power. Consult your IBM support representative to obtain an LPARCE review for your configuration.

Summary Report

This report provides a summary view of the entire systems activity over multiple measurement intervals (Figure 2-8).

- ▶ CPU Busy
- ▶ DASD rate, that is, disk I/O activity per second
- ▶ Swap rate and demand paging

R M F S U M M A R Y R E P O R T																	
PAGE 001		z/OS V1R3		SYSTEM ID SC48				START 11/17/2002-16.24.59		INTERVAL 00.04.59							
				RPT VERSION V1R2 RMF				END 11/17/2002-17.00.00		CYCLE 1.000 SECONDS							
NUMBER OF INTERVALS 7																	
DATE	TIME	INT	CPU	DASD	DASD	JOB	JOB	TSO	TSO	STC	STC	ASCH	ASCH	OMVS	OMVS	SWAP	DEMAND
MM/DD	HH.MM.SS	MM.SS	BUSY	RESP	RATE	MAX	AVE	MAX	AVE	MAX	AVE	MAX	AVE	MAX	AVE	RATE	PAGING
11/17	16.24.59	05.00	64.2	2	264.2	0	0	2	2	110	109	0	0	6	5	0.00	0.20
11/17	16.30.00	05.00	6.6	8	19.8	0	0	2	2	109	109	0	0	5	5	0.00	0.07
11/17	16.35.00	04.59	20.0	4	69.0	0	0	2	2	110	109	0	0	5	5	0.00	0.63
11/17	16.39.59	05.00	9.5	4	50.7	0	0	2	2	109	109	0	0	5	5	0.00	0.11
11/17	16.45.00	04.59	10.6	4	46.6	0	0	2	2	108	108	0	0	5	5	0.00	0.02
11/17	16.50.00	04.59	74.8	2	277.9	0	0	2	2	109	108	0	0	5	5	0.00	0.09
11/17	16.54.59	05.00	90.2	2	328.3	0	0	2	2	109	109	0	0	5	5	0.00	0.23

Figure 2-8 Summary Report

When you know your average system statistics, it is a very useful report to quickly spot unusual CPU, DASD or paging behavior.

CPU Activity Report

C P U A C T I V I T Y														
z/OS V1R3		SYSTEM ID SC48				DATE 11/17/2002		INTERVAL 05.00.509						
				RPT VERSION V1R2 RMF				TIME 16.54.59		CYCLE 1.000 SECONDS				
CPU 2064 MODEL 2C7														
CPU	ONLINE	TIME	LPAR	BUSY	MVS	BUSY	CPU	SERIAL	I/O	TOTAL	%	I/O	INTERRUPTS	
NUMBER	PERCENTAGE	TIME	PERC	TIME	PERC	NUMBER	NUMBER	NUMBER	INTERRUPT	RATE	HANDLED	VIA	TPI	
0	100.00		90.17		95.15		0B0ECB		431.3				0.75	
1	100.00		90.16		95.13		1B0ECB		429.6				0.78	
TOTAL/AVERAGE			90.16		95.14				861.0				0.76	
SYSTEM ADDRESS SPACE ANALYSIS														
				SAMPLES = 301				DISTRIBUTION OF QUEUE LENGTHS (%)						
TYPE	MIN	MAX	AVG	0	1	2	3	4	5	6	7-8	9-10	11-12	13-14
IN														
READY	1	12	4.0	0.0	6.9	3.3	12.2	49.1	21.5	4.6	1.6	0.0	0.3	0.0

Figure 2-9 CPU Activity Report (partial)

From the CPU Activity Report, take the MVS™ BUSY TIME percentage when running in Basic mode, or the LPAR BUSY TIME Percentage when running in LPAR mode.

The value reports the percentage of time all processors were busy during the RMF measurement interval. The example in Figure 2-9 on page 47 covers the same interval as before. It confirms that the SC48 partition is running at 90.16% CPU busy.

Check also the dispatching queue. The queue we are interested in is IN READY, that is, the work that is in the system and ready to be dispatched.

In this example, no immediate CPU contention is visible. Although CPU is 90% busy, something not unusual in a z/OS environment, the IN READY queue length remains below three times the number of CPs for more than 90% of the time. Note that the IN READY queue being above three times the number of CPs may not be a problem if there are non-time-critical batch jobs running in the background.

Note: Although it is usual to talk or write about a “CPU being p% busy”, this is an abbreviated statement that has no physical reality. At any point in time, a CP only has two operational states:

- ▶ Busy, that is **100** percent busy
- ▶ Idle, that is **0** percent busy

All CPU percentages in the RMF reports are relative to the RMF measurement time interval. The CPU percentages reported express the amount of time the CPU was busy over the measurement interval. Hence, the correct way to understand the report really reads, “the CPU is 100% busy p% of the time.”

Example: A report that indicates a CPU busy 30% with a measurement interval of 10 minutes really means that the CPU has been utilized 180 seconds over the 600-second interval.

Workload reports

The RMF Workload Activity Report contains information about your workload. The interpretation of the numbers depends on whether you are reporting a workload, a service class, or a reporting class. As stated earlier in this book, we strongly recommend using reporting classes.

Enclave Report

Figure 2-10 on page 49 and Figure 2-11 on page 50 illustrate a workload report for a reporting class associated with a WebSphere workload, running in enclaves. It corresponds to the WLM definitions made to the CB subsystem.

REPORT BY: POLICY=LSA510		REPORT CLASS=WASE			
		DESCRIPTION =LSA510 WAS EBUSINESS WORKLOAD			
TRANSACTIONS	TRANS.-TIME	HHH.MM.SS.TTT	--DASD	I/O--	
AVG	2.28	ACTUAL	147	SSCHRT	1.5
MPL	2.28	EXECUTION	101	RESP	1.8
ENDED	6777	QUEUED	45	CONN	1.2
END/S	22.59	R/S AFFINITY	0	DISC	0.3
#SWAPS	0	INELIGIBLE	0	Q+PEND	0.3
EXCTD	0	CONVERSION	0	IOSQ	0.0
AVG ENC	2.28	STD DEV	201		
REM ENC	0.00				
MS ENC	0.00				

Figure 2-10 Workload activity (part 1)

- ▶ AVG is the average number of active transactions during the interval.
- ▶ MPL is the average number of transactions resident in storage during the measurement interval.
- ▶ ENDED is the number of transactions that ended during the interval, and END/S is the number of transactions that ended per second. If the reporting class is set up correctly, this is a direct measure of the application throughput as seen by WebSphere.
- ▶ AVG ENC is the average number of enclaves concurrently active at any point in time. This information may be useful to size storage requirements or system recovery aspects.
- ▶ The DASD I/O section indicates the profile of the disk activity within your workload. High values for DISC, Q+PEND, or IOSQ may indicate an elongated response time.

The SSCHRT field that indicates the disk start subchannel rate, in numbers per second. From this section, you can detect a possible delay caused by I/O activity to the disk subsystem.

By comparing this value with the DASD I/O column in the Summary Report, it is possible to quantify to what extent the WebSphere application participates in the I/O activity and possibly determine whether some system tuning actions are required.

- ▶ TRANS.-TIME contains the transaction time in HHH.MM.SS.TTT units, as seen by Workload Manager. This is from the time the transaction is placed on the server region WLM queue until the time the transaction is completed.

- ACTUAL is the actual amount of time required to complete the work submitted under the service class. This is the total response time.
- QUEUED is the average time the WebSphere transaction was delayed on the WLM queue. The time can increase under full load conditions if the number of servers in MAX_SRS is too low.
- STD DEV is the standard deviation of ACTUAL. It is a measure of variability of the data in the sample. The higher the standard deviation, the more spread-out it looks on a graph.

REPORT BY: POLICY=LSA510		REPORT CLASS=WASE					
		DESCRIPTION =LSA510 WAS EBUSINESS WORKLOAD					
--DASD I/O--		--SERVICE RATES--		PAGE-IN RATES		---STORAGE----	
SSCHRT	1.5	ABSRPTN	38580	SINGLE	0.0	AVG	0.00
RESP	1.8	TRX SERV	38580	BLOCK	0.0	TOTAL	0.00
CONN	1.2	TCB	229.7	SHARED	0.0	CENTRAL	0.00
DISC	0.3	SRB	0.0	HSP	0.0	EXPAND	0.00
Q+PEND	0.3	RCT	0.0	HSP MISS	0.0		
IOSQ	0.0	IIT	0.0	EXP SNGL	0.0	SHARED	0.00
		HST	0.0	EXP BLK	0.0		
		APPL %	76.6	EXP SHR	0.0		

Figure 2-11 Workload activity (part 2)

- ▶ Note that the STORAGE field is always zero for an enclave type report. Since enclaves are not associated with a specific address space, no storage values are reported.
- ▶ The APPL% field indicates the CPU activity incurred on behalf of all activities which are part of the enclave. It is expressed as a percentage of CP time used over the interval. Note that this represents *all* the CPU activity across all address spaces spanned by the transaction, including DB2 and CICS if the transaction contains JDBC or JCA connectors.

No activity (or response time) information is reported by WLM within the CICS assigned service class or report class.

- ▶ From the above fields, it is possible to calculate a characteristic of the workload, the average CP cost per transaction. Using APPL%, the measurement interval length expressed in milliseconds and the number of ended transactions over the interval,

$$CP_millisecPerTran = Interval_length \text{ in milliseconds} * APPL\% / 100 / ENDED$$

Example: Using the RMF fields for the report class WASE in Figure 2-10 on page 49, which identifies a WebSphere application workload, you can derive that over the measurement interval:

- 2.28 transactions were concurrently active, all of them running in enclaves.
- A total of 6777 transactions ended, which translates into an average throughput of 22.59 transactions per second.
- The average response time was 147 milliseconds, with a standard deviation of 201 ms.
- Over the measurement interval, APPL% indicates that one CP was busy 76.6% of the time to service WASE. Since the measurement interval is 5 minutes, this translates into:

$$\text{Used CP time} = 300 \text{ sec} \times .766 = 229.8 \text{ sec}$$

Over the same interval, 6777 transactions have been processed. We can derive the average CP cost in millisecond per transaction:

$$\begin{aligned} \text{CP_MillisecPerTran} &= 229.8 \times 1000 / 6777 \\ \text{CP_MillisecPerTran} &= 33.90 \text{ ms} \end{aligned}$$

Address space report

As recommended, server address space activity—that does not run under an enclave—should be assigned to a service class in the STC group.

If you also defined a report class, obtain a workload report for the server region.

REPORT BY: POLICY=LSA510		REPORT CLASS=WASS			
		DESCRIPTION =LSA510 WAS SERVER AS ACTIVITY			
TRANSACTIONS		--SERVICE RATES--	PAGE-IN RATES		----STORAGE----
AVG	2.00	ABSRPTN 181961	SINGLE	0.0	AVG 56146.9
MPL	2.00	TRX SERV 181961	BLOCK	0.0	TOTAL 112293
ENDED	0	TCB 8.0	SHARED	0.0	CENTRAL 112293
END/S	0.00	SRB 0.3	HSP	0.0	EXPAND 0.00
#SWAPS	0	RCT 0.0	HSP MISS	0.0	
EXCTD	0	IIT 0.0	EXP SNGL	0.0	SHARED 3216.83
AVG ENC	0.00	HST 0.0	EXP BLK	0.0	
REM ENC	0.00	APPL % 2.8	EXP SHR	0.0	
MS ENC	0.00				

Figure 2-12 Workload report for WebSphere server address space (partial)

There are three major differences in the interpretation of the data, since the reported activity is address-space based:

- ▶ The TRANSACTION AVG indicates the number of server region address spaces active over the interval. Using this field, you can monitor the evolution of the number of servers between the MIN_SRS and MAX_SRS settings.
- ▶ STORAGE values are now filled in.
- ▶ Under normal conditions, the APPL% is typically very low. However, gradual increase in APPL% may be an indication of excessive garbage collector activity caused by a heap size too small, or a memory leak.

Using workload definitions it is possible to calculate the system uncaptured percentage value. This is the part of CP resources used by system-related services on behalf of the workloads but not directly accounted for in the enclave or address space activity.

1. For each member in the Sysplex, multiply the CPU_Busy% obtained from the CPU report by the number of CPs available to the z/OS logical partition. This brings the percentage value to a unit consistent with the APPL% reported in the workload report. Then, the sum for all systems participating in the sysplex is:

$$All_CP_Busy\% = \text{Sum of } [CPU_Busy\% * \text{Number of CPs}]$$

2. From the RMF Workload Activity report, obtain the total CP utilization reported for all workloads. This is indicated by the APPL% value for the policy. The report is obtained when option WLMGL(POLICY) is specified. The APPL% value for the policy represents the percentage of time any CP in the sysplex configuration was busy processing a workload defined in the WLM policy:

$$ALL_Wk1\% = \text{APPL\% from RMF Policy report}$$

3. The uncaptured CP value, expressed in percentage of CP activity over the measurement interval, is calculated by subtracting ALL_Wk1% obtained in step (2) from All_CP_Busy% calculated in step (1):

$$\text{uncaptured_CP\%} = All_CP_Busy\% - ALL_Wk1\%$$

Typically, the uncaptured CP% represents 10% to 20% of the total CP utilization.

Response time distribution

The workload report provides response times for all service class periods and response time distribution information. The response time distribution is provided per service class, for each service where a response time objective is defined. This is much more meaningful to the performance analyst than the average response time value.

2.3.3 DB2 SMF records

DB2 accounting times

The accounting report shown in Example 2-2 on page 55 is based on trace data generated by DB2. The DB2 accounting data is usually written to SMF. In that case, the SMF record type for DB2 accounting information is 101. The accounting counters come in classes and are defined as follows:

- ▶ Class 1 time is the time from the first SQL statement you issue in your application (that either triggers thread creation or re-signon) until disconnection (thread termination or re-signon of the next user). Both class 1 elapsed time and class 1 CPU time are reported. This time includes the time used by the application code, as well as the time used to execute inside DB2.
- ▶ Class 2 time is the time spent within DB2, processing SQL statements. Both class 2 elapsed time and class 2 CPU time are reported.
- ▶ Class 3 time is thread suspension time, for instance, when the application has to wait for an I/O to complete or wait for a lock, while processing SQL statements.

Analyzing DB2 accounting data

The counters present in the accounting report are the cornerstone for DB2 performance and tuning of your applications. They help you to understand your applications in terms of time spent in DB2 and also DB2 resource utilization.

A DB2PM accounting report contains averages. These averages are calculated by taking the sum of all occurrences for a specific counter and dividing the total by the number of occurrences (that is the number of accounting records processed). This number of occurrences is shown in the highlights section of the DB2 accounting report.

Here are some areas in the accounting report that provide useful insight into where time was spent:

Class 1 versus Class 2 time

Analyzing and comparing class 2 elapsed time and class 2 CPU time, with class 1 elapsed time and class 1 CPU time, allows you to understand how your application is working; how much time is spent in the application (class 1 - class 2) and how much time is spent in DB2 (class 2). When you experience a performance problem, and most of the time is spent in the application (class 1 time - class 2 time is big), there is probably no point in trying to optimize the work

done in DB2. A significant difference could indicate a problem in the application program.

The elapsed time distribution section of Example 2-2 shows that (on average) 90% of the elapsed time is spent in the application, and only 3% inside DB2. Therefore, if this application has a performance concern, it is most likely to be in the application.

Example 2-2 DB2 Performance Monitor - Accounting report

LOCATION: DB4B	DB2 PERFORMANCE MONITOR (V7)	PAGE: 1-4							
GROUP: DB2V714B	ACCOUNTING REPORT - LONG	REQUESTED FROM: NOT SPECIFIED							
MEMBER: DB4B		TO: NOT SPECIFIED							
SUBSYSTEM: DB4B	ORDER: PRIMAUTH-PLANNAME	INTERVAL FROM: 12/06/02 04:22:46.68							
DB2 VERSION: V7	SCOPE: MEMBER	TO: 12/06/02 04:57:45.33							
PRIMAUTH: CBASRU2 PLANNAME: DSNJDBC									
ELAPSED TIME DISTRIBUTION									
APPL	=====> 90%	CPU							
DB2	=> 3%	NOTACC							
SUSP	====> 7%	SUSP							
		=====> 22%							
		====> 9%							
		=====> 69%							
CLASS 2 TIME DISTRIBUTION									

AVERAGE	APPL(CL.1)	DB2 (CL.2)	IFI (CL.5)	CLASS 3 SUSPENSIONS	AVERAGE TIME	AV.EVENT	HIGHLIGHTS		
ELAPSED TIME	0.631648	0.061819	N/P	LOCK/LATCH(DB2+IRLM)	0.021572	0.87	#OCCURRENCES : 21728		
NONNESTED	0.631648	0.061819	N/A	SYNCHRON. I/O	0.020715	11.19	#ALLIEDS : 21728		
STORED PROC	0.000000	0.000000	N/A	DATABASE I/O	0.015984	10.23	#ALLIEDS DISTRIB: 0		
UDF	0.000000	0.000000	N/A	LOG WRITE I/O	0.004731	0.96	#DBATS : 0		
TRIGGER	0.000000	0.000000	N/A	OTHER READ I/O	0.000257	0.06	#DBATS DISTRIB. : 0		
				OTHER WRTE I/O	0.000094	0.01	#NO PROGRAM DATA: 21728		
CPU TIME	0.052238	0.013618	N/P	SER.TASK SWTCH	0.000017	0.00	#NORMAL TERMINAT: 21720		
AGENT	0.052238	0.013618	N/A	UPDATE COMMIT	0.000001	0.00	#ABNORMAL TERMIN: 8		
NONNESTED	0.052238	0.013618	N/P	OPEN/CLOSE	0.000000	0.00	#CP/X PARALLEL. : 0		
STORED PRC	0.000000	0.000000	N/A	SYSLGRNG REC	0.000016	0.00	#IO PARALLELISM : 0		
UDF	0.000000	0.000000	N/A	EXT/DEL/DEF	0.000000	0.00	#INCREMENT. BIND: 0		
TRIGGER	0.000000	0.000000	N/A	OTHER SERVICE	0.000000	0.00	#COMMITTS : 43447		
PAR.TASKS	0.000000	0.000000	N/A	ARC.LOG(QUIES)	0.000000	0.00	#ROLLBACKS : 6		
				ARC.LOG READ	0.000000	0.00	#SVPT REQUESTS : 0		
SUSPEND TIME	N/A	0.042656	N/A	STOR.PRC SCHED	0.000000	0.00	#SVPT RELEASE : 0		
AGENT	N/A	0.042656	N/A	UDF SCHEDULE	0.000000	0.00	#SVPT ROLLBACK : 0		
PAR.TASKS	N/A	0.000000	N/A	DRAIN LOCK	0.000000	0.00	MAX SQL CASC LVL: 0		
				CLAIM RELEASE	0.000000	0.00	UPDATE/COMMIT : 7.83		
NOT ACCOUNT.	N/A	0.005545	N/A	PAGE LATCH	0.000001	0.00	SYNCH I/O AVG. : 0.001851		
DB2 ENT/EXIT	N/A	559.87	N/A	NOTIFY MSGS	0.000000	0.00			
EN/EX-STPROC	N/A	0.00	N/A	GLOBAL CONTENTION	0.000000	0.00			
EN/EX-UDF	N/A	0.00	N/A	COMMIT PHI WRITE I/O	0.000000	0.00			
DCAPT.DESCR.	N/A	N/A	N/P	ASYNCH IXL REQUESTS	0.000000	0.00			
LOG EXTRACT.	N/A	N/A	N/P	TOTAL CLASS 3	0.042656	12.13			
.....									
NORMAL TERM.	AVERAGE	TOTAL	ABNORMAL TERM.	TOTAL	IN DOUBT	TOTAL	DRAIN/CLAIM	AVERAGE	TOTAL
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----
NEW USER	1.00	21720	APPL.PROGR. ABEND	8	APPL.PGM ABEND	0	DRAIN REQUESTS	0.00	0
DEALLOCATION	0.00	0	END OF MEMORY	0	END OF MEMORY	0	DRAIN FAILED	0.00	0
APPL.PROGR. END	0.00	0	RESOL.IN DOUBT	0	END OF TASK	0	CLAIM REQUESTS	12.94	281261
RESIGNON	0.00	0	CANCEL FORCE	0	CANCEL FORCE	0	CLAIM FAILED	0.00	0
DBAT INACTIVE	0.00	0							
RRS COMMIT	0.00	0							

Class 2 elapsed time versus Class 2 CPU time versus Class 3 time

Also, compare the class 2 elapsed time with the class 2 CPU time. A high difference might mean there is a lot of time spent doing things that don't use CPU, like for example, I/O, or being suspended waiting for a lock. In this case, look at the class 3 times, as well as the counter NOT ACCOUNT. The Class 2 distribution section in Example 2-2 shows that when the application is executing an SQL statement (class 2 time), most of the time the application is not using CPU, but is suspended, waiting inside DB2, before it can continue.

Class 3 wait times

To get some idea about why the application is suspended most of the time when running inside DB2, you can look at the class 3 wait counters. In Example 2-2, we see that most of the suspend time is because of database I/O wait time 0.016705 seconds on average, with on average 10.20 I/O's per occurrence (transaction). Note that 0.016705 is not the average time to do a single I/O operation, but the average time the transaction waits for all database I/O operations. The average time for a single I/O is shown in the highlights section as SYNCH I/O AVG. with a value of 0.001922. This is more or less 2 milliseconds per I/O, which is great.

Not accounted time

The *NOT ACCOUNT* counter is equal to the class 2 elapsed time minus the class 2 CPU time minus the sum of all class 3 suspension times. Normally this number is very small. In case it is not, it is often an indication that the system is very busy (most likely doing paging as paging is not captured by RMF as CPU time, or waiting for the CPU to become available). There can be other reasons for high NOT ACCOUNT values, but this is beyond the scope of this book. If you see high NOT ACCOUNT values and you are not in any of the cases mentioned above, it is probably a good idea to contact your IBM service representative to determine its cause.

DB2 thread reuse

A major component of a short-running transaction's execution time is normally the time it takes to set up a DB2 connection. In order to avoid that cost, DB2 has implemented a technique that allows threads to be reused. From the WebSphere side you can indicate that you want to enable this by using data sources. (Note that DB2 data source support is only available starting in DB2 V7). To verify that threads are actually reused inside DB2 by WebSphere, you can check the NEW USER field, 21720 in our case, and compare it to the total number of occurrences 21728. This shows that in almost all cases the thread was reused, which is excellent.

Tip: Although DB2 Performance Monitor still exists as a product, a new follow-on product, called DB2 Performance Expert for z/OS, is available. For more information on DB2 Performance Monitor and DB2 Performance Expert, see *DB2 Performance Monitor for z/OS*, SG24-6867.

2.3.4 WebSphere SMF records

WebSphere SMF Record Interpreter

The WebSphere for z/OS SMF Record Interpreter is a tool that enables the interpretation of complete SMF output data sets from the IBM z/OS utility program IFASMFDP. It writes a header line for all SMF record types and a detailed dump for SMF record type 120.

The tool is a Java utility. It is executed by a Java Virtual Machine (JVM) under the z/OS or OS/390 UNIX environment. The base tool can be downloaded from the WebSphere Application Server for z/OS and OS/390 Web site at:

http://www.ibm.com/software/webservers/appserv/zos_os390/index.html

The z/OS SMF Summary Viewer

We used an enhanced version of the SMF Browser, the WebSphere V. 4.0.1 for z/OS SMF Summary Viewer, made available by the Washington Systems Center. The tool can be downloaded from the Advanced Technical Support Information Web site. See the document *WP100244* in the White Papers category at:

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

This version of the SMF Browser adds a summary report, much more easy to read than the standard report. It shows activity for each J2EE server instance, bean, and method from the SMF type 120 records.

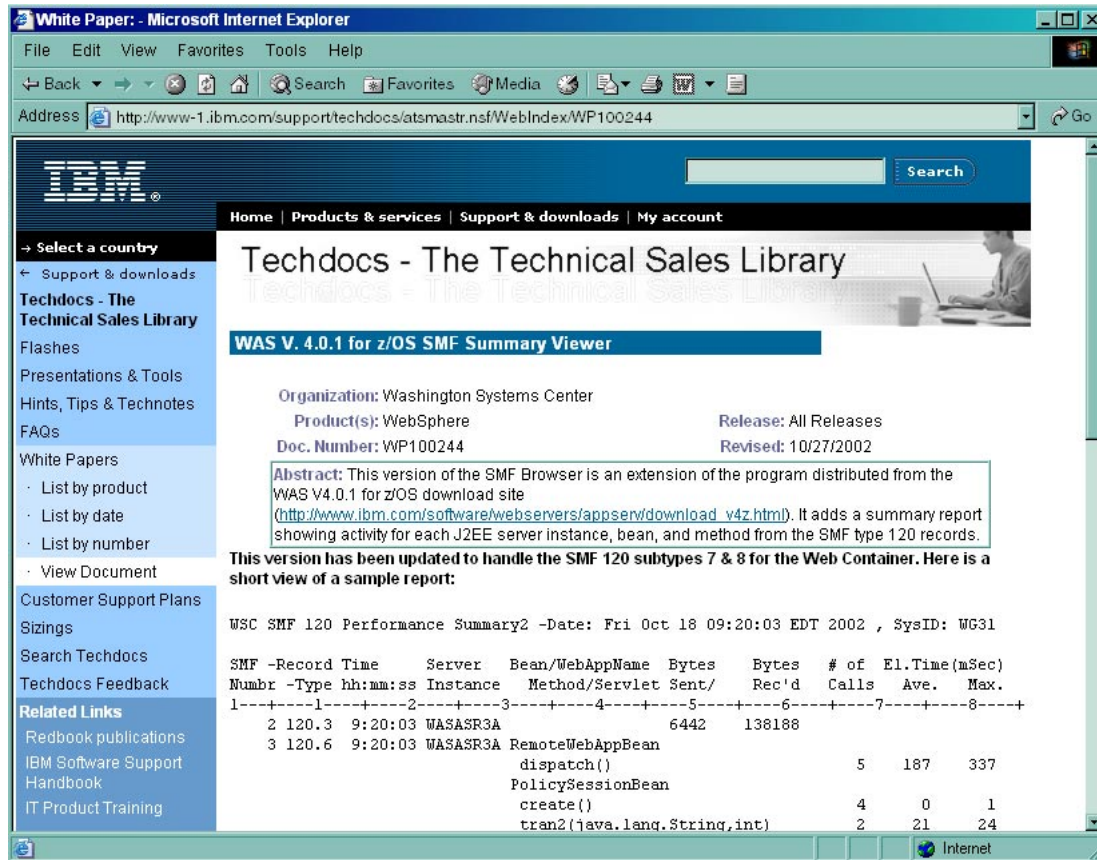


Figure 2-14 Obtaining the SMF Summary Viewer from WSC

Java class files and source files are in the WSCSMFPerf2.jar file; documentation to run the analysis program is in WSCSMF120.doc file.

Running the z/OS SMF Summary Viewer

Since the tool does not interpret any SMF records other than record 120, it is recommended that you filter out all other records. The following example dumps SMF records 120 from system data sets SYS1.SC48.MAN1 and SYS1.SC48.MAN2 into a sequential file named FRANCK.SC48T.SMF.

Example 2-3 Using IFASMFDP to copy SMF records into a sequential file

```
//LSA5101 JOB 999,'ITS0',
//          MSGCLASS=T,NOTIFY=&SYSUID,CLASS=A
//DUMP1    EXEC PGM=IFASMFDP
//INSMF1   DD DSN=SYS1.SC48.MAN1,DISP=SHR
```

```
//INSMF2 DD DSN=SYS1.SC48.MAN2,DISP=SHR
//SMFDATA DD DSN=FRANCK.SC48T.SMF,
//          DCB=(RECFM=VBS,LRECL=32760),
//          SPACE=(CYL,(25,50)),
//          UNIT=SYSALLDA,
//          DISP=(NEW,CATLG)
//*
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
          OUTDD(SMFDATA,TYPE(120))
          INDD(INSMF1,OPTIONS(DUMP))
          INDD(INSMF2,OPTIONS(DUMP))
```

The WebSphere for z/OS SMF Record Interpreter dumps all the WebSphere for z/OS-relevant data into a printable output file.

For example, to interpret data from a cataloged sequential file named FRANCK.SC48T.SMF, previously created using the IFASMFDP utility as described in Example 2-3, and send the output to file WTSCplexSMFout.txt, you would go to the TSO OMVS shell and execute the command as shown in Figure 2-15.

```

IBM
Licensed Material - Property of IBM
5694-A01 (C) Copyright IBM Corp. 1993, 2001
(C) Copyright Mortice Kern Systems, Inc., 1985, 1996.
(C) Copyright Software Development Group, University of Waterloo, 1989.
All Rights Reserved.
U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
Disclosure restricted by GSA-ADP schedule contract with IBM Corp.
IBM is a registered trademark of the IBM Corp.
-----
Set up environment variables for Java and Servlets for OS/390 -
-----
PATH reset to ./usr/lpp/java/IBM/J1.3/bin:/usr/lpp/Printsrv/bin:/bin:.
PATH is /var/iwl/bin:./usr/lpp/java/IBM/J1.3/bin:/usr/lpp/Printsrv/bin:/bin
:./u/franck:
FRANCK:/u/franck: >

==> java -cp WSCSMFPerf2.jar com.ibm.ws390.sm.smfview.Interpreter
"FRANCK.SC52A.SMF" ./WTSCplexSMFsummary.txt 1>WTSCplexSMFout.txt
INPUT
ESC=¢ 1=Help      2=SubCmd    3=HlpRetrn  4=Top       5=Bottom    6=TSO
       7=BackScr   8=Scro1l   9=NextSess 10=Refresh 11=FwdRetr 12=Retrieve

```

Figure 2-15 Running the SMF Record Interpreter tool

The summary report of the z/OS SMF Record Interpreter would be saved in file WTSCplexSMFsummary.txt and available to be browsed or edited through ISPF.

Sample output

Data from the sequential file is produced record by record. Each record contains a number of triplets, which are first described in the record's header section (the first part of a record). The description is then followed by the triplet contents, which are presented by the tool in the sequence of their appearance within the record.

The WebSphere for z/OS SMF Summary Viewer interprets each section in its specific way and prints the interpreted data into the output file.

1. Summary report file sample from Trade2

The detail report file lists each activity that occurs in the server during the collection interval, in server, Web container, and J2EE container. A sample report is shown in Example 2-4.

Example 2-4 SMF Browser (1)

WSC SMF 120 Performance Summary2 -Date: Sun Nov 10 13:37:00 EST 2002 , SysID: SC52

SMF Nubr	-Record -Type	Time hh:mm:ss	Server Instance	Bean/WebAppName Method/Servlet	Bytes Sent	Bytes Rec'd	# of Calls	EI. Time(mSec)	Ave.	Max.
30	120.1	13:37:00	FMISRVC		579	4191				
31	120.5	13:37:00	FMISRVC	Trade_WebApp dispatch()			1	1	1	
32	120.7	13:37:00	FMISRVC	/welcome.jsp				1		
				JSP 1.1 Processor				1		
				trade Web Application_0						
33	120.1	13:37:00	FMISRVC		863	4559				
34	120.5	13:37:00	FMISRVC	TradeRegistryBean findByPrimaryKey(trade.Registr >ejbLoad >ejbActivate login(java.lang.String) >ejbStore >ejbPassivate			1 1 1 1 1 1	1 0 0 0 0 0	1 0 0 0 0 0	
				TradeAccountBean findByPrimaryKey(trade.Account >ejbLoad >ejbActivate getBalance() >ejbPassivate			1 1 1 1 1	3 0 0 0 0	3 0 0 0 0	
				Trade_WebApp dispatch()			1	16	16	
				TradeSession create() login(java.lang.String,java.la getBalance(java.lang.String)			2 1 1	0 1 3	0 1 3	
35	120.7	13:37:00	FMISRVC	/tradehome.jsp				1		
				TradeAppServlet				15		
				JSP 1.1 Processor				1		
				trade Web Application_0						

This trace shows the activities in server, J2EE container, and Web container caused by Login transaction in Trade2 sample at a specific time. It includes invocation of welcome.jsp, tradehome.jsp and TradeAppServlet by the Web container, and EJB activities such as each method invocation of TradeRegistryBean entity EJB, TradeAccountBean entity EJB and TradeSession session EJB. Response time of each method call and number of bytes downstream and upstream served by the server are also collected.

2. Summary report file sample from eITSO application

The summary report displays statistic data, such as average and maximum elapsed time of the server, container, Web container, and J2EE container for each type of activity during the collection interval. A type of activity can be the same JSP invocation, or the same method call on the same EJB. The following is a sample of a summary report file for an interval of 5 minutes.

Example 2-5 SMF Browser (2)

WSC SMF 120 Performance Summary2 -Date: Mon Nov 18 18:15:03 EST 2002 , SysID: SC50

SMF Numbr	-Record Type	Time hh:mm:ss	Server Instance	Bean/WebAppName Method/Servlet	Bytes Sent	Bytes Rec'd	# of Calls	El.Time(mSec) Ave.	Max.
44	120.3	19:30:01	FMESRVB		34004	226469			
45	120.6	19:30:01	FMESRVB	ItemEntity findByPrimaryKey(itemEntityPac			5	1553	2129
				WebERWWNO_WebApp create()			3	1	1
				driveLoadServlet(java.lang.Str			1	1345	1345
				dispatch()			8	3587	19361
				WarehouseEntity findByPrimaryKey(warehouseEnti			17	3428	7107
				WebERWWJustPC_WebApp create()			4	0	0
				driveLoadServlet(java.lang.Str			2	992	1002
				dispatch()			8	11524	49825
				PriceChangeSession create()			5	17	45
				priceChangeSession(priceChange			5	3639	6930
				PaySession create()			15	1	2
				paySession(paySessionPackage.P			15	18521	46859
				DeliverySession deliverySession(deliverySessio			1	46129	46129
				create()			2	2	2
				RemoteWebContainer create()			14	0	2
				driveLoadServlet(java.lang.Str			14	526	1361
46	120.6	19:30:01	FMESRVB	WebERWWD_WebApp create()			4	1	1
				driveLoadServlet(java.lang.Str			2	751	1344
				dispatch()			2	29765	59506
				NewOrderSession create()			2	0	0
				NewOrderEntity findByWIdAndDId(short,short,bo			7	6667	32528
				WebERWWPY1_WebApp create()			12	0	1
				driveLoadServlet(java.lang.Str			9	270	1346

47	120.8	19:30:01	FMESRVB	dispatch()	26	12730	82043
				WebERWWS_15			
				WebERWWSL_17			
				WebERWWPQ_16			
				WebERWWjmsPRR_25			
				eRWWPriceChangeHTTPSession_26			
				WebERWWPC_21			
				DEController	1	59328	59328
				SimpleFileServlet	1	22	22
				JSP 1.1 Processor	1	59326	59326
				/DEAGResults.jsp	1	54922	54922
				WebERWWDelivery_20			
				SimpleFileServlet	8	27	51
				WebERWWNO_19			
				SimpleFileServlet	3	49	61
				/error.jsp	5	132	168
				JSP 1.1 Processor	5	132	168
				WebERWWJustPC_14			
				PAYController	8	33396	62242
				/PAYAGResults.jsp	8	31355	57871
				SimpleFileServlet	18	25	67
				/error.jsp	8	680	898
				JSP 1.1 Processor	8	33361	62177
				WebERWWPay_24			
49	120.3	19:35:01	FMESRVB		57124	147311	

For example, we can see that `findByPrimaryKey` method on `ItemEntity` EJB was called 5 times with an average elapsed time of 1553 ms and maximum elapsed time of 2129 ms. Another example is `SimpleFileServlet`, which is responsible for serving static pages in the Web app. The report shows the number of `SimpleFileServlet` calls in each Web app and the average elapsed time in the Web container.

Observation - enabling SMF record type 120

We tried to run a set of measurements to evaluate the overhead brought by SMF recording of SMF record type 120 in our environment. Since we did not run in a controlled environment, the absolute values for each may be imprecise and there is no guarantee that you would find exactly the same result in your installation, but the difference between the two runs provides at least an indication.

The measurements were made on one of the z/OS partitions, system SC48, running two shared CPs on a zSeries 900 Model 1C8. This test was run with WebSphere V4.0.1 at maintenance level W401407.

SMF 120 records were enabled through the SMEUI, and no subtype selection was specified in SMFPRMxx.

Activity Records

The test stopped short because the SMF data sets were filled in approximately 2 minutes, although they were sized to support approximately one hour. Since our RMF measurement interval was set to 5 minutes, we were never able to obtain data to quantify the observation.

Although we could not quantify, our recommendation after this test is to disable SMF recording for type 120 Activity Records on a production system. If type 120 Activity Records are required, they should be produced on a test or development system.

Interval Records

We ran a similar test, now with the interval records enabled. The recording interval in the SMEUI was set to zero in order to synchronize on the SMF interval and obtain data consistent with other SMF records.

Two measurements were made, one with 50 clients and the other with 100 clients. All results were obtained from RMF summary and workload reports. Table 2-1 on page 64 summarizes the results obtained for the 50 clients workload.

Table 2-1 SMF test at 50 clients

At 50 clients (7 tran/sec)	SMF disabled	SMF enabled	[disabled - enabled]
Total CP%	35.3%	38.2%	2.9%
Tran/sec	7.0	7.0	0
MPL	0.37	0.47	0.10
WebSphere workload CP APPL %	20.5%	22.4%	1.9%
Transaction Response Time in ms	52	66	14 (26%)
WebSphere workload CP APPL ms per tran	29.3	32.0	2.7

Observations:

- ▶ The throughput remains unchanged, at 7.0 transactions per second.

- ▶ The transaction response time is significantly increased; this is also reflected in the MPL.
- ▶ No significant change is visible in the RMF reports for I/O activity or memory usage.
- ▶ The total CPU cost is expressed in percentage of one shared CP in the two-way partition (this test was run on a zSeries 2064-1C8). It represents 2.9% of CP time over the measured interval, of which 1.9% are reflected within the WebSphere workload. This translates into:
 - 0.27% of CP time per transaction over the measured interval reported as an addition to the workload part
 - 0.14% of CP time per transaction for the non-enclave-related CP part, that is, the control region address space, system and subsystem activity, plus the uncaptured time.

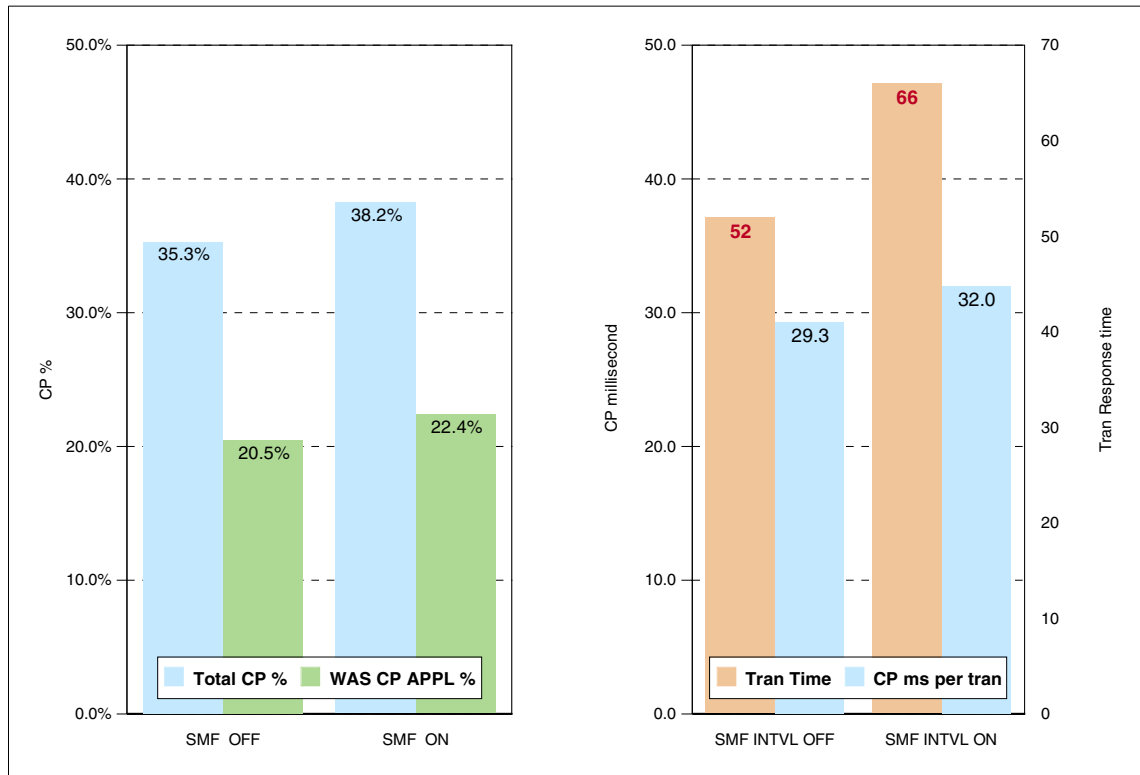


Figure 2-16 SMF type 120 Interval records ON or OFF for 50 clients

Results consistent with the above observations were observed when the load was increased to 100 clients:

- ▶ There is again an increase in response time; this is also reflected in the MPL.

- ▶ The total cost is 5.8% of CP time over the measured interval, of which 3.9% are reflected within the WebSphere workload. This translates into:
 - 0.28% of CP time per transaction added to the workload part.
 - 0.15% of CP time per transaction for the non-enclave CP part.

Table 2-2 SMF test at 100 clients

At 100 clients (13.8 tran/sec)	SMF disabled	SMF enabled	[disabled - enabled]
Total CP%	58.2%	64.0%	5.8%
Tran/sec	13.8	13.8	0
MPL	0.83	1.09	0.26
WebSphere Workload CP APPL %	40.8%	44.7%	3.9%
Transaction Response Time in ms	60	79	19 (30%)
WebSphere Workload CP APPL ms per tran	29.5	32.5	2.7

As expected, it is impossible to produce and record performance data without actually impacting the performance of what you are trying to measure. Every software measurement tool represents a certain cost, in terms of CPU overhead, memory used, and input/output activity.

Based on the above observations, our recommendations are:

- ▶ In the absence of another WebSphere application monitoring tool (see Part 2, “WebSphere performance tools” on page 107), SMF type 120 records are the only source of information on application activity. If you require application performance information on the Web or J2EE container, enable SMF type 120 interval record. Be aware that SMF recording has a price, but running a production system with no information on application activity may prove even more costly.
- ▶ Do not enable recording of SMF type 120 Activity Records on a production system, unless you have a compelling reason to do so. When doing so, review the size and placement of SMF data sets for I/O performance.
- ▶ Be selective. Only enable SMF type 120 interval records when you intend to use them at some point in time.
- ▶ When SMF type 120 Interval records are enabled:
 - If you plan to use an online reporting tool that exploits SMF records type 120, set the recording interval to the value recommended by the tool.

- In the absence of any other recommendation, synchronize the recording interval with the SMF and RMF measurement interval (specify a value of zero in the SMEUI). This will simplify the analysis when trying to correlate application data from SMF type 120 with RMF data.

2.3.5 Garbage Collection (GC) trace

WebSphere runs applications in a JVM. When this JVM fails to allocate memory due to a shortage of Java heap, it starts garbage collection. To identify whether 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 SMUEI 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. Example 2-6 shows a GC trace output.

Example 2-6 GC Trace

```
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>
```

```

    <GC(64): refs: soft 0 (age >= 32), weak 0, final 6210, phantom 0>
    <AF[64]: completed in 187 ms>

    <AF[65]: Allocation Failure. need 32784 bytes, 2327 ms since last AF>
    <AF[65]: managing allocation failure, action=1 (28967432/131004928)
    (1104624/3145728)>
    <GC(65): GC cycle started Thu Dec  5 15:55:18 2002
    <GC(65): freed 40895480 bytes, 52% free (70967536/134150656), in 185 ms>
    <GC(65): mark: 163 ms, sweep: 22 ms, compact: 0 ms>
    <GC(65): refs: soft 0 (age >= 32), weak 0, final 6264, phantom 0>
    <AF[65]: completed in 186 ms>
    .....
    <AF[178]: Allocation Failure. need 32784 bytes, 1571 ms since last AF>
    <AF[178]: managing allocation failure, action=2 (187840/134150656)>
    <GC(178): GC cycle started Thu Dec  5 16:00:25 2002
    <GC(178): freed 38656688 bytes, 28% free (38844528/134150656), in 218 ms>
    <GC(178): mark: 204 ms, sweep: 14 ms, compact: 0 ms>
    <GC(178): refs: soft 0 (age >= 32), weak 0, final 7774, phantom 0>
    <AF[178]: completed in 219 ms>

    <AF[179]: Allocation Failure. need 4112 bytes, 1756 ms since last AF>
    <AF[179]: managing allocation failure, action=2 (0/134150656)>
    <GC(179): mark stack overflow>
    <GC(179): GC cycle started Thu Dec  5 16:00:27 2002
    <GC(179): freed 38779872 bytes, 28% free (38779872/134150656), in 279 ms>
    <GC(179): mark: 265 ms, sweep: 14 ms, compact: 0 ms>
    <GC(179): refs: soft 0 (age >= 32), weak 0, final 6843, phantom 0>
    <AF[179]: completed in 279 ms>
    .....
    <AF[466]: Allocation Failure. need 32784 bytes, 426 ms since last AF>
    <AF[466]: managing allocation failure, action=2 (1588480/134150656)>
    <GC(596): GC cycle started Thu Dec  5 16:05:16 2002
    <GC(596): freed 7924000 bytes, 7% free (9512480/134150656), in 181 ms>
    <GC(596): mark: 169 ms, sweep: 12 ms, compact: 0 ms>
    <GC(596): refs: soft 0 (age >= 32), weak 0, final 1200, phantom 0>
    <AF[466]: managing allocation failure, action=3 (9512480/134150656)>
    <AF[466]: managing allocation failure, action=4 (9512480/134150656)>
    <AF[466]: clearing all remaining soft refs>
    <GC(597): GC cycle started Thu Dec  5 16:05:17 2002
    <GC(597): freed 18312 bytes, 7% free (9530792/134150656), in 196 ms>
    <GC(597): mark: 182 ms, sweep: 14 ms, compact: 0 ms>
    <GC(597): refs: soft 2 (age >= 32), weak 0, final 0, phantom 0>
    <GC(598): GC cycle started Thu Dec  5 16:05:17 2002
    <GC(598): freed 5544 bytes, 7% free (9536336/134150656), in 199 ms>
    <GC(598): mark: 186 ms, sweep: 13 ms, compact: 0 ms>
    <GC(598): refs: soft 0 (age >= 32), weak 0, final 0, phantom 0>
    <AF[466]: completed in 580 ms>

```

AF[x] indicates the xth time memory allocation failed, and GC(y) indicates the yth garbage collection since the server region started. The % free number in each GC trace line indicates how much free memory is available in the JVM after the GC. If this number decreases over a period of time, it means there is 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 a 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, this is set in directive `JVM_HEAPSIZE=xxx` in the `current.env` file.

You can plot a chart from the verbose GC trace for easier analysis. There is an awk script sample provided at:

<http://www-1.ibm.com/support/techdocs/atmastr.nsf/WebIndex/TD100748>

This awk script formats a trace file to a semicolon-delimited condensed file that can be directly imported to spreadsheet tools like Microsoft Excel or Lotus® 1-2-3®.

The following chart was generated by the trace shown earlier. The chart shows there was a JVM memory problem with the application. The problem indications include the increasing frequency of memory allocation failures and garbage collection, less and less memory collected by GC, and lastly, the steadily decreasing free memory percentage in JVM. Indeed, this trace was generated by an application with intended memory leak.

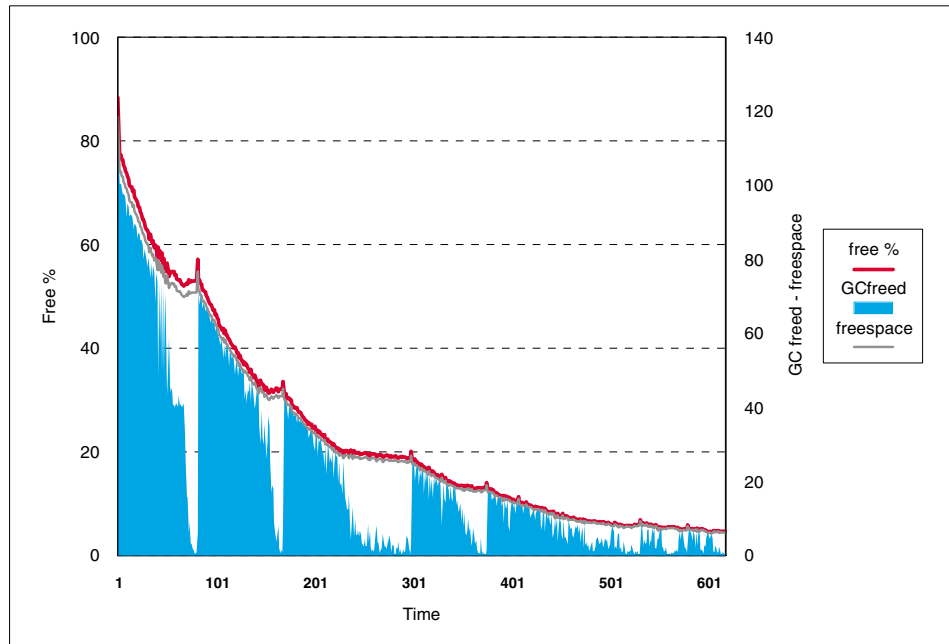


Figure 2-17 Plotting Garbage Collection trace

2.4 Establishing the diagnosis

2.4.1 Overview

Here we introduce a general approach to troubleshooting a WebSphere application performance problem in a production environment. The intention is to identify the cause of the problem. The actual solution depends on the problem itself.

Performance analysis is an iterative process, and you should be prepared to go through the process a number of times. The correction you apply trying to remove the bottleneck may not produce the improvements you are looking for, or may just move you from one problem to another further down the line.

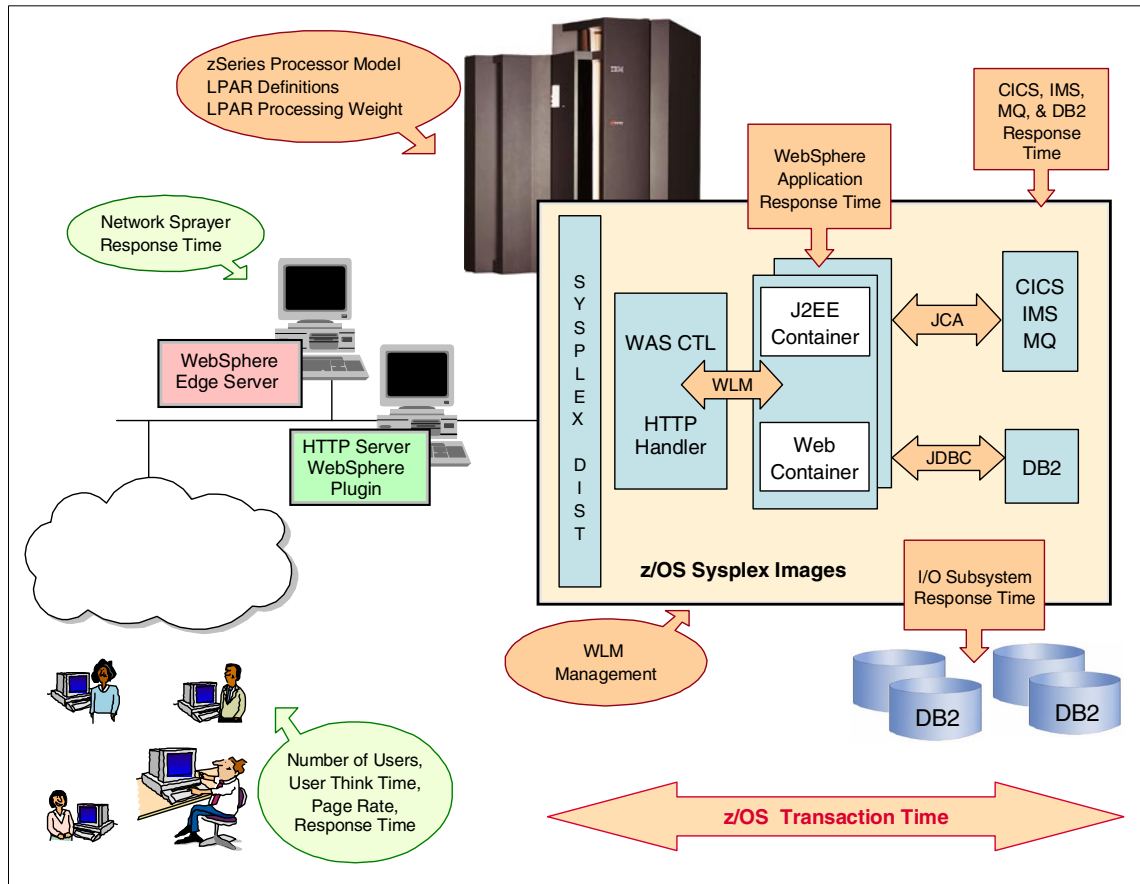


Figure 2-18 Performance monitoring - an overview

- ▶ Understand the performance expectations of your system resources and applications.
- ▶ Take a quick snapshot view of the system
 - If something is extremely out of line with experience, investigate it immediately.
 - If something is only moderately out of line, remember it and continue to the next step.
- ▶ Other quick checks and diagnostics
 - See “Other quick checks and diagnostics” on page 74 for suggestions.
- ▶ Look at the logical user requests that are not meeting response time expectations.

- If the logical user request response times within WebSphere server regions are ok, check components before server regions. This is mostly network-related and is outside the scope of this redbook.
- If the user request response times within WebSphere server regions are not satisfactory, examine these user requests more closely and continue.
- Map the logical user requests into WebSphere transactions. To the extent that you can't map user requests to WebSphere transactions, you may need to guess and make assumptions.
- Segregate the WebSphere transactions into sets of good and bad transactions based on response times.
- Examine the resources used in each component of all bad transactions and identify common features and/or anomalies.
- Form hypotheses that explain 80% of the observations of good and bad transaction sets.
- Test these hypotheses, one by one, by gathering additional information.
- ▶ Be prepared to repeat this process when the identified problem has been resolved.

2.4.2 Initial diagnostics

Understand the expectations

In an ideal world, you would have a detailed, documented Service Level Agreement that has been derived from accurate capacity planning and confirmed by detailed historical monitoring data.

Reality may be different. The means by which performance expectations may be derived are discussed in 2.1.1, "Setting your performance expectations" on page 28. The answer may be "I don't know." Be aware that the less you understand your performance expectations, the murkier the problem will seem and the harder it will be to identify it. At various points in the process you are required to make a judgement.

To decide whether a number on a report is good or bad for your application with your combination of hardware and software and your business priorities is virtually impossible unless you have a clear understanding of what the data means and what is acceptable in your environment.

Performance analysis has no simple point-and-click solution. The tool that can do this for you has not yet been invented. You have to make judgements based on experience, rules of thumb, or guesswork in order to progress. These judgements may sometimes be wrong and lead you in a wrong direction.

Quantify: Take a quick snapshot view of the system

Performance issues, especially when originated from user or management complaints, tend to generate strong feelings, even frustrations. In such cases, the best is not to participate in the turmoil but to gather factual information to quantify the problem.

- ▶ If something is extremely out of line with experience, investigate it immediately.
- ▶ If something is only moderately out of line, remember it and continue to the next step.

The first thing one needs to check is the hardware and software environment at the system level:

1. LPAR processing weights

This is to check that the LPAR is receiving the expected level of hardware resources. This will need to be checked against documented norms for your installation. It is possible that a change has been implemented incorrectly resulting in less service to the LPAR you are interested in. There may have been a deliberate change to assign resources to another LPAR for business reasons.

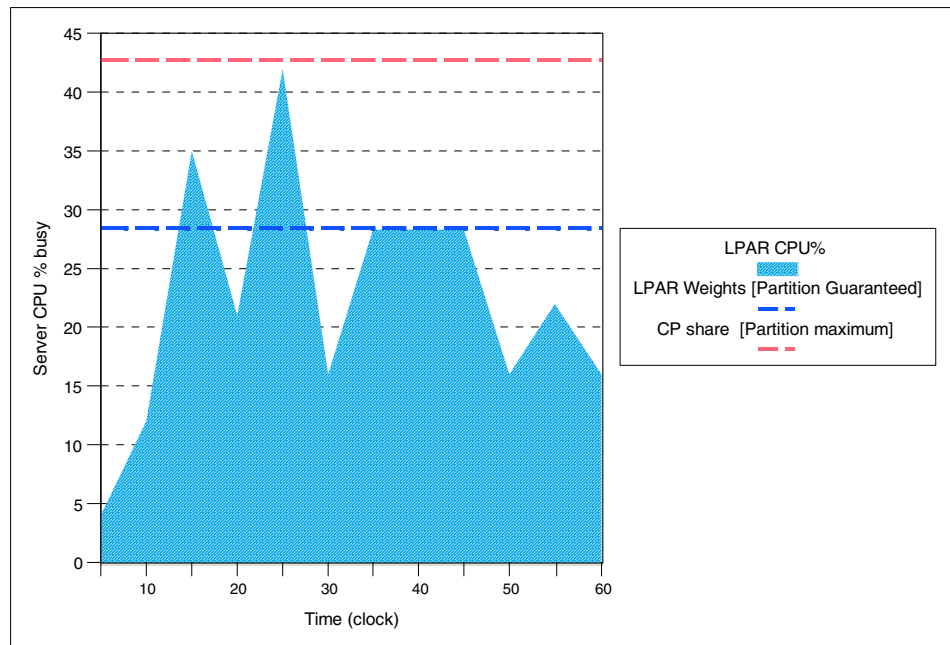


Figure 2-19 Partition view from Partition Data Report

Use the RMF Partition Data Report to check the used LPAR CPU usage against the guaranteed and maximum share available to your partition.

Figure 2-19 shows an example of LPAR CPU activity over 60 minutes, and the guaranteed and maximum share. For intervals 35, 40 and 45 it is highly probable that the partition is CPU-constrained to its guaranteed share because of activity in other logical partitions. Although not an anomaly, this is something you should remember for the rest of the analysis.

Remember that your LPAR CPU share is relative to the sum of the weights of all partitions. As a consequence, your guaranteed share is reevaluated for every change in the logical configuration:

- Every time a logical partition is activated or deactivated
- Every time operations update the processing weights
- Dynamically if your logical partition participates in an LPAR cluster

The Partition Data Report alone cannot tell you whether this is good, bad, normal or not, but it can tell whether it fits your expectations.

2. CPU Queue

In the distributed world, running CPU above 50% is unusual. On zSeries, running CPU at 90% or more is not necessarily an indication of a problem and is, indeed, common. Even 100% is not necessarily a problem; in this case you need to investigate further to evaluate how much queuing it causes. The CPU Activity Report will help you.

Check the Queuing report in the CPU Report. If the queue length substantially exceeds three times the number of CPs online in the configuration, one workload may have a CPU delay problem. Although it may not be a performance problem and may only affect a non-priority batch workload, you should remember it for a later step.

3. Paging activity

- a. Check the system paging level in the RMF summary report. The RMF Summary Report indicates the system wide demand paging rate. As with CPU, high paging is not necessarily a problem, but high paging may lead to a CPU penalty and response time problems.
- b. If system paging is indicated, then check if paging occurs at the server region level. Check the STORAGE and PAGING sections in the workload report for the server regions. Make sure you check the address space, *not* the enclave, since the enclave will show zero values for PAGING and STORAGE.

Other quick checks and diagnostics

- ▶ Have you followed all the recommended performance tuning guidelines?

WebSphere 4.0.1 tuning recommendations can be found in chapters 9 - 10 in *WebSphere Application Server V4.0.1 for z/OS and OS/390: Operations and Administration*, SA22-7835. If using the IBM HTTP Server, tuning recommendations can be found at:

<http://www.ibm.com/software/webservers/httpservers/doc/v51/2tabcont.htm>

► Contention

If WebSphere has to contend for resources, its performance may degrade. To check for some form of contention in your system, enter the z/OS command:

```
D GRS,C
```

The result will need interpretation. During the course of the residency, we encountered a problem due to contention within RRS. Output from the previous command was:

```
S=SYSTEMS SYSZATR WTSCPLX1-RESTART
SYSNAME      JOBNAME      ASID      TCBADDR    EXC/SHR    STATUS
SC42         RRS          03EE      007EC788  EXCLUSIVE  OWN
SC50         RRS          002A      007EBAA0  EXCLUSIVE  WAIT
SC52         RRS          002A      007EAD90  EXCLUSIVE  WAIT
SC48         RRS          03EF      007EB8B0  EXCLUSIVE  WAIT
NO REQUESTS PENDING FOR ISGLOCK STRUCTURE
NO LATCH CONTENTION EXISTS
```

Systems SC50, SC52 and SC48, where we were running our WebSphere Servers, were waiting for system SC42 to release a lock.

► WLM definitions

See 2.2, “Workload Manager controls” on page 33.

► Environment Variables in current.env for each server instance

These may also be inspected by checking the STC output. Detailed descriptions can be found in *WebSphere Application Server V4.0.1 for z/OS and OS/390: Assembling J2EE Applications*, SA22-7836.

- JVM_HEAPSIZE - This is the Java heap size. It is only worth checking if you know what the value should be. This needs to be big enough for the application to run without Garbage Collection becoming too much of an overhead and small enough not to present too big an overhead to the z/OS image. There is no rule of thumb. The recommendation is to use the default.
- MAX_SRS and MAX_SRS may constrain the maximum number of server regions that can be started by WLM. A reasonable MAX_SRS starting value is three times the number of CPs available in the z/OS image.

Still, MAX_SRS should be at least as large as the number of different service classes that might be used by transactions run in the server as explained in, “Managing the number of application server regions” on page 33.

- JAVA_COMPILER - Ensure that this is not set.
- JVM_DEBUG - Ensure this is not set.
- TRACEALL - Set to 0 or 1 (0 preferred). If set to 1, note that tracing to SYSPRINT causes a much bigger overhead. This can be avoided by setting TRACEBUFFLOC=BUFFER.
- TRACEBASIC - Ensure this is not set.
- TRACEDetail - Ensure this is not set.
- ▶ Some configuration settings defined in the SMEUI are stored in DB2, but not reflected in current.env. Check that:
 - Production server is set to YES.
 - Debugger allowed is set to NO.
 - Server region SMF Activity Records is not activated.
- ▶ Check for unnecessary tracing. Turn off unless required.

- Component tracing

To find out which component trace options are active, issue this MVS command from a console or SDSF:

```
/D TRACE,COMP=ALL
```

The output typically consists of a number of pages. Check that all traces are either MIN or OFF unless there is a definite requirement to the contrary.

The entries for WebSphere Application Server will look something like this:

```
-----
SYSBBOSS      ON          HEAD    18
  ASIDS       *NOT SUPPORTED*
  JOBNAMES    *NOT SUPPORTED*
  OPTIONS     MINIMUM
  WRITER      BBOWTR
-----
```

To turn tracing off for any given component, use this command:

```
TRACE CT™,OFF,COMP=xxxxxxx
```

where xxxxxx is the name of the component.

- JRAS tracing - set in the file identified by `com.ibm.ws390.trace.settings` in the `jvm.properties` file for the Server Instance.

To turn all JRAS tracing off, specify:

```
*=all=disabled
```

- JDBC tracing - is turned on and off by entries in the file identified by the `DB2SQLJPROPERTIES` environment variable, which is in `current.env` for the Server Instance by the SMEUI. For example:

```
# Any lines starting with the pound sign '#'
```

```

# are comments. Please see the DB2 for OS/390
# Application Programming Guide and Reference
# for Java for the description of these settings.
#
DB2SQLJSSID=DB4B
DB2SQLJATTACHTYPE=RRSAF
DB2SQLJMULTICONTEXT=YES
DB2CURSORHOLD=NO
# The following items are for connection pooling.
db2.connpool.max.size=5
db2.connpool.idle.timeout=600
db2.connpool.connect.create.timeout=0
#DB2SQLJPLANNAME=DSNJDBC
#DB2SQLJ_TRACE_FILENAME=/tmp/mytrc

```

JDBC tracing is activated by setting variable DB2SQLJ_TRACE_FILENAME to any value. When the line is commented out, tracing is disabled.

– SMF record 92 subtypes 10 and 11

SMF type 92 records provide information about HFS activity. Subtype 10 is written at file open and Subtype 11 at file close. Given that WebSphere will require a lot of HFS file open and close activity as part of normal operation, collecting these subtypes is likely to impose a significant overhead.

- ▶ Check your joblog for CEEDUMPs or SVC dumps.
- ▶ Hardware faults - There will usually be indications in SYSLOG in the form of error or warning messages if there is some kind of hardware fault.
- ▶ Check your WebSphere error log for any indications of failures. If WebSphere is experiencing abends, the error recovery (the restart of a server region) will severely impact performance. You may also find that the scheduling environment status becomes STOPPED, which will prevent new server regions being started. The status of the APPLENV can be checked by:

```
D WLM,APPLENV=<APPLENV name>
```

If you don't know the name for a given application server, try displaying all the APPLENVs on the system:

```
D WLM,APPLENV=*
```

To reset the status of a given APPLENV:

```
V WLM,APPLENV=<APPLENV name>,RESUME
```

Note that this may not always be sufficient to restart a failing server. If you are in a situation where a control region has started, but has failed to start a server region due to a stopped applenv and resuming it does not cause a server region to start, you may have to restart the affected control region.

It is important to try to understand what is causing the abends and to resolve the problem. Problem Determination is not covered in this redbook. Refer to *WebSphere for z/OS V4 Problem Determination*, SG24-6880.

2.4.3 Where does it hurt?

Now that you have an overview of the system behavior, let's check the WebSphere workload. Whenever possible, try to quantify the characteristics of the workload:

- ▶ Throughput, expressed in transactions per second obtained from the workload report
- ▶ CP usage, both total CPU% from the Summary or CPU report, and WebSphere applications from APPL% in the workload report
- ▶ Response time, either AVG from the Workload report, or preferably 90th percentile evaluated from the response time distribution report

Figure 2-20 shows two typical examples of what you should expect. On a z/OS production system, total CP usage is typically in the range 90 to 100%. This is not a problem. Note that all CPU percentages have been normalized to reflect the capacity of the whole sysplex as explained in 2.3.2, "RMF reports" on page 39.

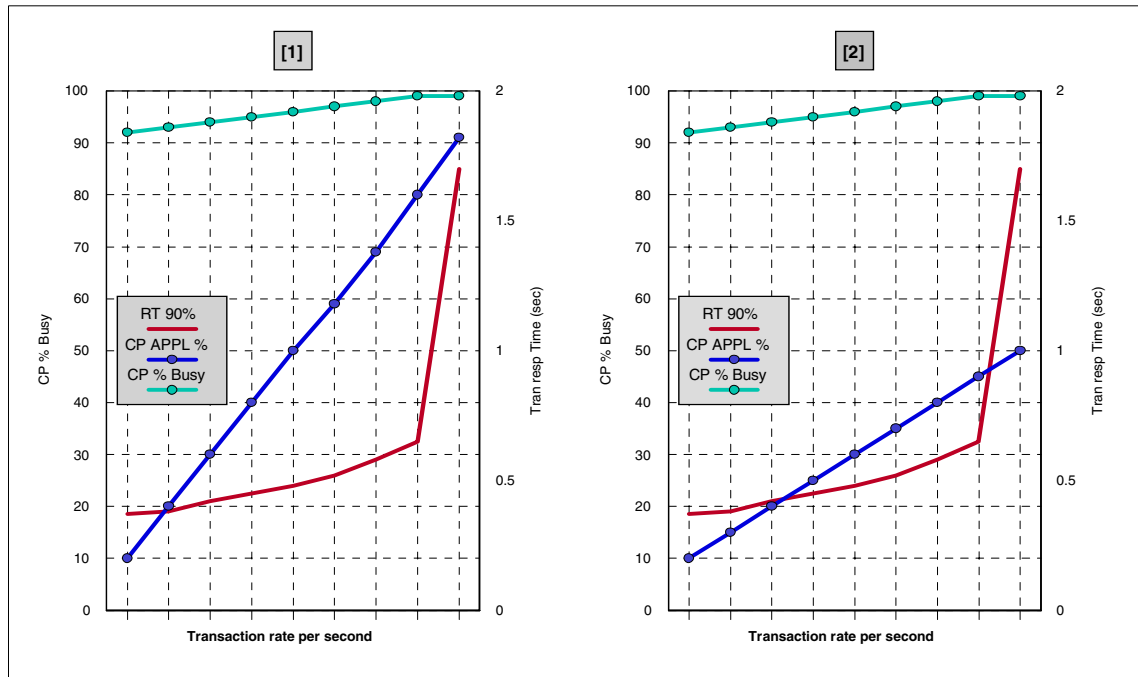


Figure 2-20 CP usage, response time, and throughput

In both examples, the workload CP APPL% grows almost linearly with the transaction rate. This is what should be expected when no problem is present. Although visibility may vary with the length of the measurement interval, it is very likely that in case of a performance issue CP usage and throughput will not correlate in a linear fashion.

Graph [1] in Figure 2-20 illustrates a no-problem situation. The system behaves as expected in the range observed, even though the amount of CPU resource used may not meet your expectations.

The 90th percentile response time remains sub-second until the workload CP usage reaches 90%, where there is an important increase. This is normal.

Graph [2] illustrates a typical throughput problem.

The knee of the response time curve appears long before the APPL% CP usage reaches 90%. More investigation is required to determine the cause of the problem.

1. Check the Workload Manager definitions. Other workloads running on the sysplex and competing for CP resource may take precedence over WebSphere applications. If this is by mistake, change the WLM settings. If

this is desired, WLM is enforcing business priorities as defined and it is no longer an issue that has a technical solution.

2. One resource, other than the CP resource, is constrained when the workload increases. It may be another z/OS-managed physical resource (I/O or storage) or a logical resource. If a logical resource, it may be within the WebSphere infrastructure, within another z/OS component (DB2, CICS, IMS), or within the application itself.

Figure 2-21 illustrates another practical example using a WebSphere application workload running dedicated on a single z/OS image. It comes from one of our test and was run on partition SC48 with 2 online CPs (hence, the 200% on the CP% busy axis) on zSeries model 1C8.

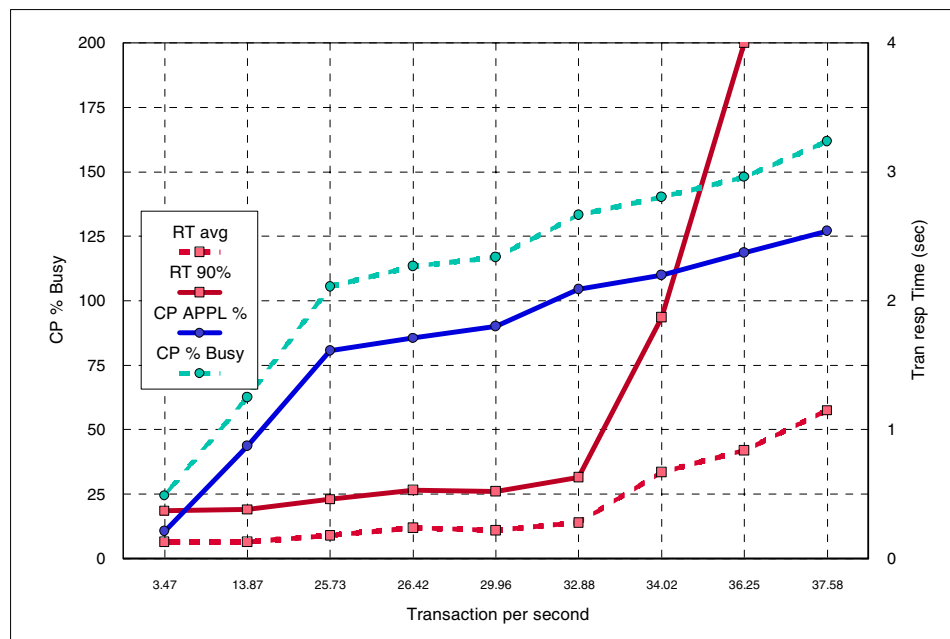


Figure 2-21 CPU% and response time versus throughput

It shows what you should expect from such an analysis for a WebSphere workload, when no memory problem is present:

- ▶ The CPU usage from APPL% plots in a linear way with the throughput. In the above example, we used a linear regression with a 0.99 R-square.

When the workload APPL% does not plot in a linear fashion, this is usually an indication of a performance problem.

- ▶ The response time does *NOT* plot linear. It slowly grows up to a point where it brutally jumps. The knee of the curve indicates the scalability limit of the workload given the current logical and physical configuration.

In this example, the knee of the response time curve appears just above 32 transactions/second while the APPL% is approximately 110% and the total LPAR CPU% is 140%.

From the graph in Figure 2-21 on page 80, we can deduce that:

- ▶ There is a response time problem above 32 transaction per second.
- ▶ It is *not* related to a CPU usage into the WebSphere application server.
- ▶ It is *not* related to a CPU queuing problem at the server level since the server has not reached the LPAR guaranteed share.
- ▶ It is most likely that we do not have a memory problem.

Another indicator that may prove useful is the average CP usage per transaction. It may be expressed in various unit. In this document we will use the number of milliseconds of CP per transaction.

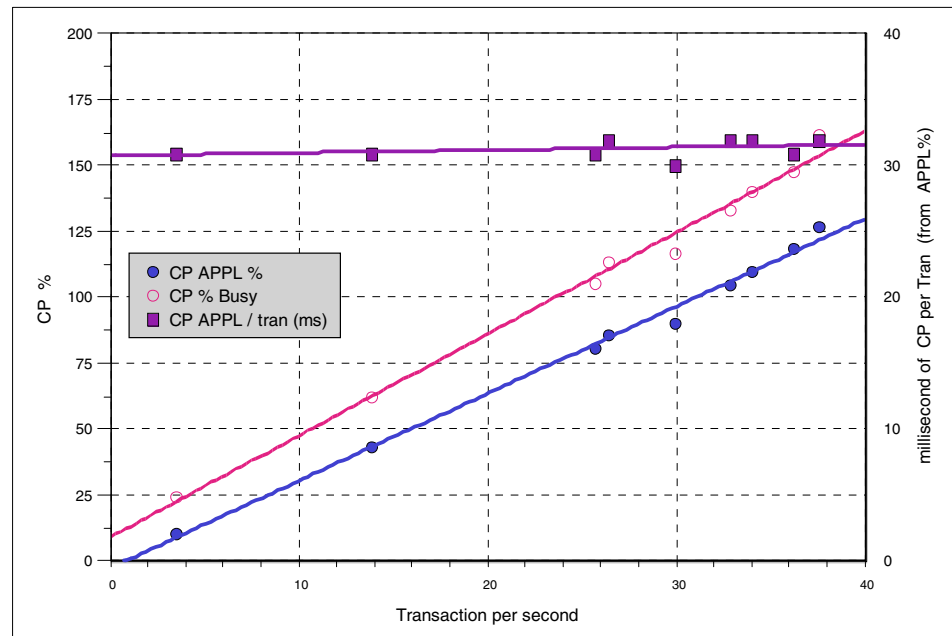


Figure 2-22 CP time per transaction

Under normal circumstances the average CP millisecond per transaction should be nearly constant across the throughput range, as shown in Figure 2-22. A significant variation is usually an indication of a performance problem.

You can quantify the CP per transaction using three methods, the only important point being that the interpretation of the numbers should be kept consistent with the method chosen:

- ▶ The workload CP APPL%, that is, considering only the workload reported in the enclave, which includes application CP time in WebSphere and in any subsystem (DB2, MQ, IMS or CICS) called on behalf of the transaction.
- ▶ The workload CP APPL% plus the WebSphere server address spaces. The time will then reflect variations when additional servers are started/stopped because of the Application Environment or when servers are recycled. It will also show the time incurred because of the Garbage Collector.
- ▶ The total CP time, that is, the workload CP APPL% plus the WebSphere server address spaces plus the apportioned uncaptured CP time. Although this is the gross value preferred for cost calculations, it may not be the best one to use for performance analysis.

Once the performance problem is qualified, you can follow the path to the next step. We suggest three specialized paths:

- ▶ Do you suspect a memory leak or a heap problem?
If CPU consumption in the server address space (not the application environment) is higher than expected, go to 2.4.4, “Check for memory problem” on page 82.
- ▶ Are you experiencing a delay problem?
If response time gets significantly worse (getting to the knee of the curve) as you apply more load without using available CPU, go to 2.4.5, “The delay pain” on page 83.
- ▶ Do you consume more CPU than expected?
If CPU utilization seems higher than expected for the current transaction rate, go to 2.4.6, “The CPU pain” on page 85.

2.4.4 Check for memory problem

Although it should never happen in a production environment, a memory leak is one of the first issues you should investigate if you have any suspicion. There are two reasons for that approach:

- ▶ A memory leak will induce both CPU and delay problems. Unless you have a specialized tool described in Part 2, “WebSphere performance tools” on

page 107, reporting values will be distorted by the problem and you will be unable to follow a performance path.

- ▶ A memory leak affects the server region address space, hence the availability of the WebSphere infrastructure.

Run a garbage collection trace

If you suspect a memory leak, you may first wish to run a verbose GC trace as explained in 2.3.5, “Garbage Collection (GC) trace” on page 67.

- ▶ If the garbage collection analysis confirms a memory leak, there is very little that can be done in the production environment. Send the faulty application back to development for correction.
- ▶ Check for correct Java heap size. To do so, check the percentage of time spent in GC processing. A good rule of thumb is that you should spend less than 5% of your time in Java GC processing.

Select a GC cycle after your application has run long enough to reach a steady state. Repeat this test for a number of GC cycles after that.

- Locate the time since the last allocation failure for this GC: “YYYY ms since last AF”.
- Locate the time it took to complete the GC processing: “completed in XXX ms”.
- Estimate the % of time in GC processing:
 - If GC processing is less than 5% of the time, then your heap size is OK.
$$\text{"completed in XXX ms" / "YYYY ms since last AF"} < 5\%$$
 - If you are spending 5% or more of your time in Java GC, increase your heap size. Then check your Java GC activity again. Note, however, that if you are on a storage constrained system, increasing the Java heap to reduce GC overhead may result in more paging.

2.4.5 The delay pain

Check requests not meeting response time expectations

If the logical user request response times within WebSphere server regions are OK, check components before server regions, mostly network-related components.

If the user request response times within WebSphere server regions are not OK, examine these user requests more closely and continue.

- ▶ Map the logical user requests into WebSphere transactions. To the extent that you can’t map user requests to WebSphere transactions, you need to guess and make assumptions.

- ▶ Segregate the WebSphere transactions into sets of good and bad transactions based on response times.
- ▶ Examine the resources used in each component of all bad transactions and identify common features and/or anomalies.
- ▶ Form hypotheses that explain 80% of the observations of good and bad transaction sets.
- ▶ Test hypotheses, one by one, by gathering additional information.
One hypothesis could be that the code is written badly. Do not attempt to use profilers in production. Send the code back to a test system for profiling.
- ▶ Then do it all again.

Is the delay in the WebSphere application or somewhere else?

- ▶ Apply enough load to drive up the response time.
Work your way down the transaction path until you reach a place where response time is good. Using this approach, you can locate the source of your delay.

In the RMF Monitor I Workload Activity report, check the response time in the Application Environment for your WebSphere application.

- If the server region response time is good, you probably have a delay in the network. Fix any network problems and if you still have performance issues, start the diagnosis process again from the top.
- If you are storing data in DB2, CICS, or IMS, check your response time using the appropriate database monitor (DB2PM, CICS statistics, IMS pars). See 2.3.3, “DB2 SMF records” on page 54 for an example of DB2PM.
- If the database response time is also bad, do normal database tuning to improve the response time. Then, if you still have performance issues, start the diagnosis process again from the top.
- If the server region response time is also bad and the database response time is good, you have a delay in the server region. Go to the next step.
- ▶ Collect SMF 120 interval records to help locate the delay.

To enable this record, select “Write SMF Interval Records” in the SMEUI. You also must enable collection of this record in the SMFPRMxx PARMLIB member.

Looking at a summary view of your SMF 120 data, you can see what beans and methods are experiencing poor performance. You may need help from your application developer to understand the cause of the problem.

In some cases, SMF 120 data will not provide information that is low-level enough to isolate the problem. For example, activity in servlets, JSPs, and regular Java classes called by these servlets and JSPs is accumulated under method, dispatch, bean, RemoteWebAppBean.

2.4.6 The CPU pain

Are you consuming CPU in the WebSphere application server?

- ▶ Collect RMF Monitor I, including a Workload Activity Report.
- ▶ Check if the WebSphere application is really the source of your CPU activity, or if you are consuming CPU somewhere else.
- ▶ Locate the APPL% value for the Application Environment associated with your WebSphere application. Calculate the CPU cost per transaction.
- ▶ Compare the WebSphere APPL% value with the APPL% for the whole system and with the APPL% value from other report classes on the system.
- ▶ Check the system uncaptured time for any unusual value.
- ▶ If the problem is really in the WebSphere application, continue to the next step.
- ▶ Collect SMF 120 interval records to help locate the beans and methods with lots of CPU activity.
 - Select “Write SMF Interval Records” in the SMEUI if not already enabled.
 - Looking at a summary view of your SMF 120 data, you should be able to locate which beans and methods are experiencing poor performance. If you are not familiar with the application, you may need help from your application developer to understand the cause of the problem. Remember that these records only report on elapsed time, so beans with calls to DB2, CICS, or IMS might show long elapsed times though they only make a small contribution to the CPU consumption in the server region.
 - In some cases, SMF 120 data will not provide information that is low-level enough to isolate the problem. For example, activity in servlets, JSPs, and regular Java classes called by these servlets and JSPs is accumulated under method, dispatch, bean, RemoteWebAppBean. For lower-level analysis of where you have delays, some other tool may be needed.
 - In some cases it may be necessary to profile your application. This is something that should not be done on a production system; send the application back to development for further testing. Check WebSphere Studio Application Developer or other Java profiling tools, available from IBM or other vendors. At the time of this writing the Jinsight profiling tool can be obtained from the IBM AlphaWorks Web site. For more information, see:

<http://www.alphaworks.ibm.com/formula/jinsight>

Check the technical Sales Library Web site (Techdocs) for a white paper on WebSphere for z/OS application debugging and profiling. See:

<http://www-1.ibm.com/support/techdocs/atmastr.nsf/WebIndex/WP100250>



The ITSO test environment

In this chapter we describe the environment under which we ran our performance tests, and the examples that we used to exercise the available monitoring tools.

3.1 Hardware and software configuration

Our servers, applications, and network access were configured for high availability and load distribution, in a manner similar to that which we might expect to find in a production environment. In addition to this, we had to make provision for:

- ▶ A server to generate scripts and feed them into the WebSphere servers
- ▶ Servers on which the application monitoring tools ran

The configuration was designed in accordance with all the high availability principles described in the redbook *Enabling High Availability e-Business on zSeries*, SG24-6850.

3.1.1 The sysplex configuration

The three systems we used in our performance tests were all logical partitions in a 2064 zSeries server. Due to the fact that the hardware configuration changed during the duration of the project, the reports associated with each example may show an IBM 2064 model 1C7, or 2C7, or 1C8.

Each LPAR was allocated 384 MB of real storage. The actual number of CPs and the processing power assigned to each LPAR (processing weights) varied during the project, based on each test's requirements and contention with other concurrent projects.

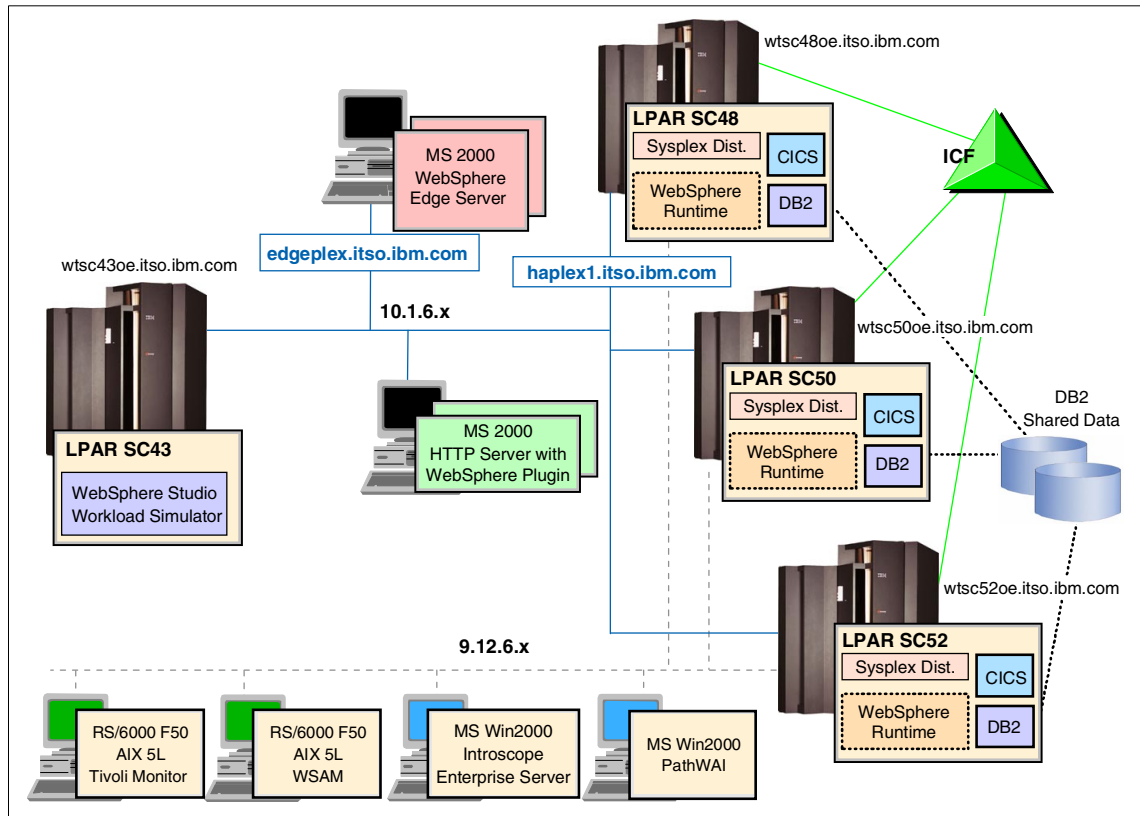


Figure 3-1 The ITSO configuration used

The following hardware and software components are relevant for the high availability of WebSphere for z/OS. See Table 3-1 on page 90 for the release levels of the products we used.

- ▶ Coupling Facility - Two external Coupling Facilities (CF) are installed.
- ▶ Open System Adapter - For networking connections, we used two Open System Adapter (OSA) adapters.
- ▶ WebSphere Application Server V 4.0.1 for z/OS.
- ▶ TCP/IP with Sysplex Distributor.
- ▶ Resource Access Control Facility (RACF) uses a sysplex-wide shared database.
- ▶ Resource Recovery System (RRS) is used for two-phase commit.
- ▶ Automatic Restart Manager (ARM) is set up to start all necessary components on the same system in case of component failure. If the entire

z/OS image fails, ARM restarts DB2 and the WebSphere Daemon and System Management address space on another system to release the locks.

- ▶ Workload Manager (WLM) is set up in goal mode.
- ▶ Lightweight Directory Access Protocol server (LDAP) is set up to run in sysplex mode and in TDBM mode.
- ▶ DB2 V 7.1 runs in data sharing mode.
- ▶ CICS V 2.2.

A detailed description of the setup of each of these components is available in the redbook *Enabling High Availability e-Business on zSeries*, SG24-6850.

Table 3-1 Product levels used

Product name	Release
z/OS	V1.3 at RSU0208
WebSphere	V4.0.1, level W401400 or level W401407
DB2	V7.1 at RSU0208 JDBC driver at UQ69569 level
CICS	V2.2
CICS Transaction gateway	V5.0

The setup of the WebSphere Application Servers is shown in Figure 3-2 on page 91. Our test sysplex comprised three z/OS instances named SC48, SC50 and SC52.

WebSphere Application was at maintenance level W401400, as shown in Example 3-1. The information is printed on the log when the server is started.

Example 3-1 WebSphere 4.0.1 maintenance level

```
BB0J0011I JVM Build is "J2RE 1.3.1 IBM OS/390 Persistent Reusable VM
build hm131s-20020723 (JIT enabled: jitc)".
BB0U0245I CURRENT CB SERVICE LEVEL IS build level W401400 release
cb401_serv date 09/19/02 16:31:13.
```

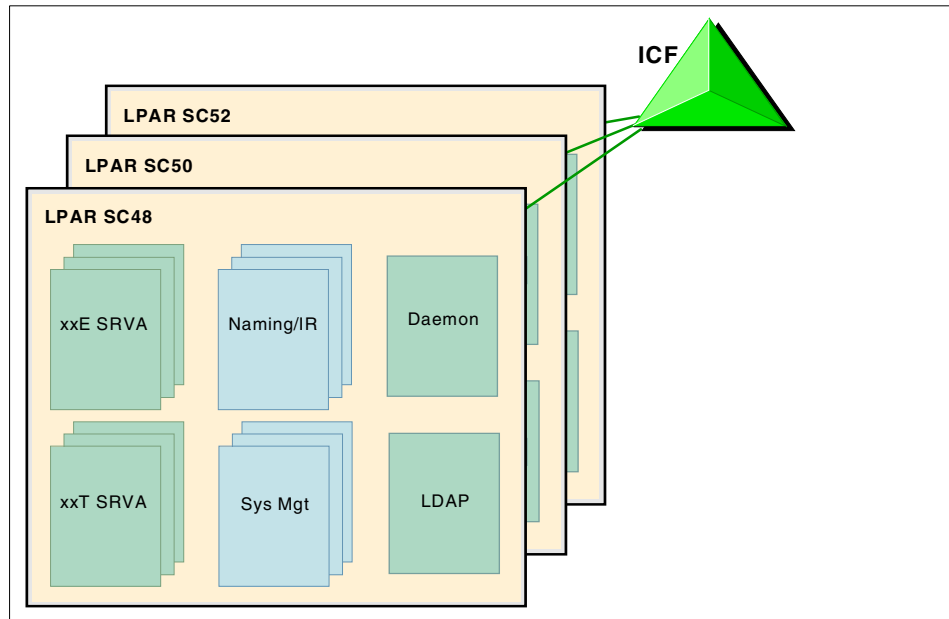


Figure 3-2 WebSphere Application setup at ITS0

The set of WebSphere servers had different names depending on which monitoring tool was being tested at the time, so we describe them here using generic names. We had two servers running in the sysplex, named xxESRV and xxTSRV.

Table 3-2 WebSphere servers in the configuration

z/OS Image	Server Name	J2EE Application
SC48	xxESRVA	eITSO
	xxTSRVA	PRR Trade2
SC50	xxESRVB	eITSO
	xxTSRVB	PRR Trade2
SC52	xxESRVC	eITSO
	xxTSRVC	PRR Trade2

Each WebSphere server has three instances, one on each z/OS image, all named according to the same convention. Thus the three instances of xxTSRV were xxTSRVA on SC48, xxTSRVB on SC50, and xxTSRVC on SC52.

The xxESRV servers ran workloads called eITSO and PRR, while the xxTSRV servers ran a workload called Trade2.

Table 3-3 WebSphere server name for each monitor

Monitor	xx Prefix	Server Name	J2EE Application
Introscope	IN	INESRVA, INTSRVB, INTSRVC	eITSO
		INTSRVA, INTSRVB, INTSRVC	PRR Trade2
PathWAI	OM	OMESRVA, OMESRVB, OMESRVC	eITSO
		OMTSRVA, OMTSRVB, OMTSRVC	PRR Trade2
WebSphere Studio	WS	WSESRVA, WSESRVB, WSESRVC	eITSO
Application Monitor		WSTSRVA, WSTSRVB, WSTSRVC	PRR Trade2

The workloads were developed by IBM for test purposes, and are designed to exercise a wide variety of WebSphere functions, including access to CICS and DB2. These workloads are further described in 3.1.3, “ITSO test workloads” on page 96.

Depending on the purpose of the test, the number of server regions was adjusted using the MIN_SRS and MAX_SRS parameters. Each instance had between two and eight server regions to do its work.

In addition, each z/OS also ran instances of the usual WebSphere support servers: Daemon, Systems Management, LDAP, Naming, and Interface Repository.

3.1.2 Network access

Clients

On the client side, the requirement was to permit load distribution across the server instances, while ensuring that:

- ▶ The right server was selected for eITSO or Trade2 or PRR.

- ▶ Any required affinity was preserved, bypassing the load distribution while the client needed to communicate with a specific server instance for the duration of a session.

To achieve this, we set up the network access to the WebSphere sysplex as shown in Figure 3-3 on page 93. We used a combination of the following to provide the necessary mixture of availability and load distribution:

- ▶ Sysplex distributor on z/OS
- ▶ The IBM HTTP Server running under Windows 2000 on a PC 300® PL
- ▶ The WebSphere Edge Server, also on a Windows 2000 PC 300 PL

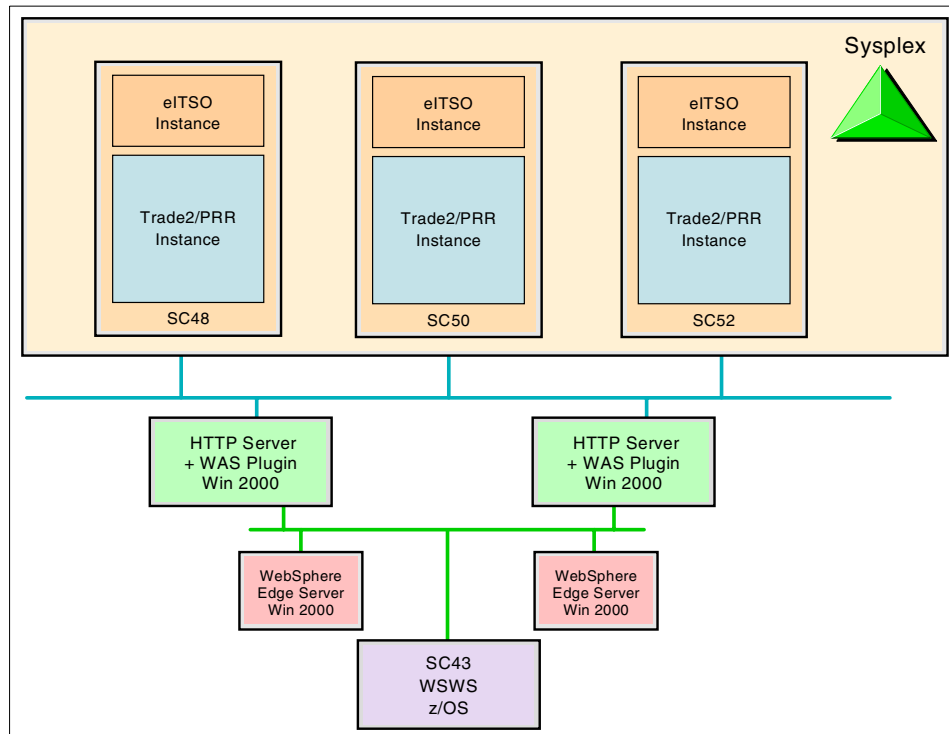


Figure 3-3 Network access from clients

The way it works is as follows:

SC43 is a z/OS system running WebSphere Studio Workload Simulator (WSWS), which executes browser scripts and sends them across the network to the desired target server. We used WSWS to simulate up to 1000 browser clients accessing the WebSphere applications.

The WebSphere Studio Workload Simulator scripts are all configured with the same server name, edgeplex.itso.ibm.com, but use different TCP port numbers

to distinguish between applications: eITSO and Trade2 or PRR servers. For example, we might have the xxTSRV instances all listening on port 7070 and the xxESRV instances all listening on port 7080.

SC43 /etc./hosts file maps edgeplex.itso.ibm.com to the cluster address of the WebSphere Edge Servers. Therefore, all requests sent out by WebSphere Studio Workload Simulator go to the active WebSphere Edge Server instance. In a high availability configuration such as this, only one WebSphere Edge Server instance responds to the cluster address while the other simply monitors the availability of its partner.

WebSphere Edge Server is configured to forward all packets for the cluster address, and the TCP ports for which it is configured, to one of two HTTP servers. Thus, all HTTP requests from WebSphere Studio Workload Simulator reach an HTTP Server, where the TCP connection is terminated.

The clever bit comes in the plug-in running in the HTTP Server. The plug-in's job is to inspect all incoming requests and to determine whether to give them to the local HTTP Server, or to forward them to another server, in this case WebSphere on z/OS. The plug-in is set up to work as follows:

- ▶ Incoming requests are identified by target port number alone. The port number is used to determine which of the two servers the request must be sent to.
- ▶ The other check that the plug-in makes is to look for the JSESSIONID keyword in any cookie. If this keyword identifies a particular WebSphere server instance, it indicates a session affinity and this request goes to that instance.

So, if there is an affinity the request is sent (now on a new TCP connection) to an IP address that identifies a server instance uniquely. In our case we used a static VIPA defined on each TCP/IP stack.

If there is no affinity, the request is sent to an IP address that represents the Sysplex Distributor. We set up SC48 as the primary distributing stack, so the request goes there first. SC48 identifies which z/OS instances can service the request—all of them, because all have applications listening on the appropriate ports—and uses WLM to make a decision about where the connection should go.

Note that, at the time the tests were run, Sysplex Distributor could only distribute four ports per target IP address (distributed VIPA). Since we had three tools to try out and two servers for each tool, we required at least six ports. Therefore, we

defined one distributed IP address for the eITSO servers and one for the Trade2/PRR servers.

Note: This restriction is to be lifted by the fix to PQ65205, which increases the number of ports from 4 to 64.

In its way back from WebSphere, the response to the HTTP request goes back to the HTTP Server, where it is passed through back to WebSphere Studio Workload Simulator without any further processing.

This setup differs from the way it would usually work in production. In real life, the incoming requests would be distinguished by URI and the plug-in would translate specific portions of the URI to the appropriate port numbers. Thus, the context root in the URI could determine which WebSphere server would handle the request. We did it this way to make it easier for Workload Simulator to drive the traffic patterns needed.

Some of the most important definitions we used to set up the network access can be seen in Appendix B, “Configuration files” on page 291. These include the HTTP server definitions and the z/OS TCP/IP definitions.

Monitoring tools

To keep the test network “clean” and isolated from other work, such as our own TSO access and other projects’ traffic, we ran two separate TCP/IP stacks in each z/OS instance, as shown in Figure 3-4.

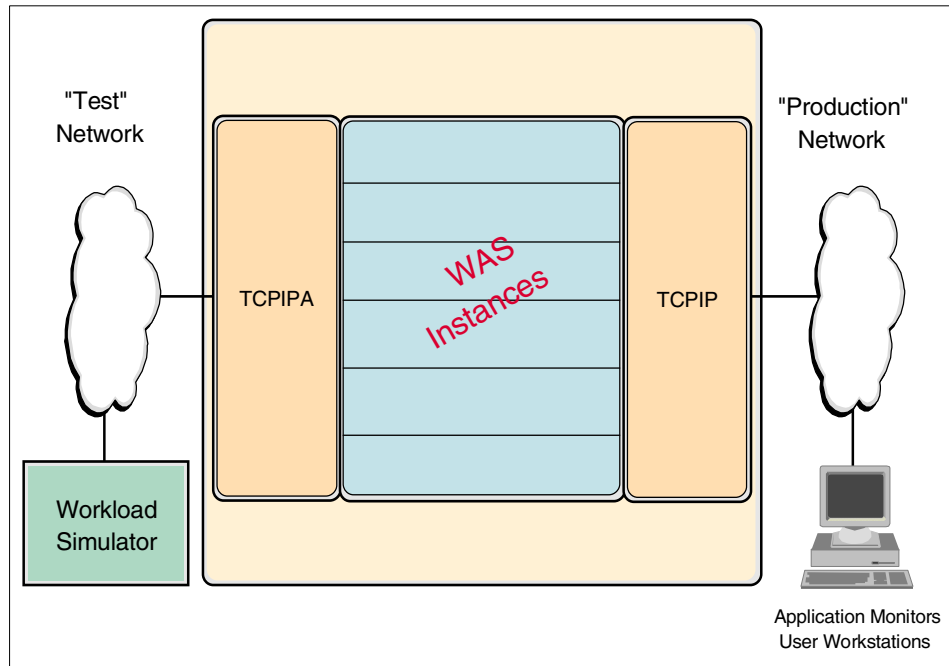


Figure 3-4 Network access from monitors

All the application monitoring servers, as well as the workstations used to access them via browsers, were connected to the production side of the network.

Since WebSphere is a well-behaved UNIX System Services application, it connects to both stacks in the common INET environment, and we were able to access the servers from both the Workload simulator test network and from the application monitors on the production network. Using two stacks in this manner is not always easy, or even possible. For example, WebSphere issues a `gethostid()` call to USS to obtain its IP address. This address is sent to the WSAM monitor, which then sets up further TCP connections to it. We had to make sure that the address obtained came from the production stack—the test stack addresses are unreachable from the production network.

3.1.3 ITSO test workloads

We used the following applications to create workloads for our tests:

- ▶ Trade2
- ▶ eITSO (based on a modified version of eRWW)
- ▶ PRR

The eRWW and PRR workloads are IBM internal workloads. They were developed for test purposes, and are designed to exercise a wide variety of WebSphere functions including access to CICS, IMS, MQSeries®, and DB2.

Trade2

The Trade application models an online brokerage application, providing Web-based services such as login, buy, sell, get quote, and more. It uses a servlet to drive a session EJB that calls a data bean that uses container-managed persistence to return data to a JSP that generates the HTML returned to the user; the general flow is illustrated in Figure 3-5.

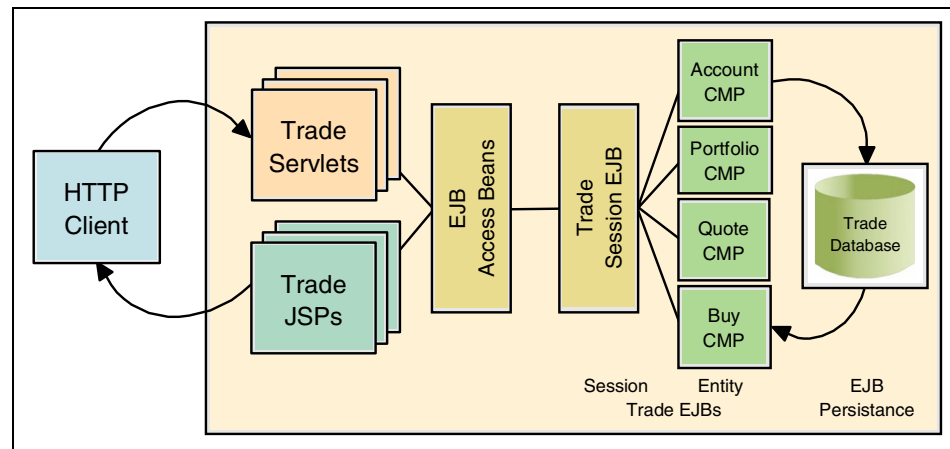


Figure 3-5 Components and flow within Trade2

Trade2 requires that users log in before trading, and uses standard Java functions to create and manage session objects.

The Web container for Trade2 specifies HTTP sessions, so WebSphere will create a JSESSIONID cookie. The Web container also specifies that the session objects are not to be saved to DB2, so session affinity to the same WebSphere application server region is required. To accomplish this, WebSphere adds a CloneID to the end of the JSESSIONID cookie so that requests can be routed back to the same system they originated from.

For the tests, we used a special Trade2 servlet called TradeScenarioServlet. This facility was created explicitly for use by workload generation programs, and is designed to be called repeatedly. On each call it randomly calls one Trade2 function. If no session is yet established, then it calls the logon function. Subsequent calls use that session until the servlet randomly decides to log off the user.

More information on the Trade2 workload is available at:

http://www-3.ibm.com/software/webservers/appserv/wpbs_download.html

PRR

The PRR application was developed by IBM. It is a front end to specialized workloads.

It was used to drive a stateless EJB session bean to call a CICS transaction, using the JCA CICS connector, CICS TG 5.0, to read or write DB2 data. The EJB then returns the commarea data to be formatted by a JSP.

eITSO

The eITSO application is a modified version of the eRWW application, a workload that was developed by IBM to model an order management system.

3.1.4 WebSphere Studio Workload Simulator

WebSphere Studio Workload Simulator was used to drive the applications to load the system and test the monitoring techniques and tools. The tool captures the output of interactions between a Web browser and a server, then replays those same interactions later.

WebSphere Studio Workload Simulator contains two main components, the controller and the engine, which are described as follows:

- ▶ The controller

This component runs on a workstation and is used to capture and modify the test script. Test scripts are FTP'ed to the engine machine for execution there. During execution, the controller connects to port 3000 on the engine machine to display a monitor of the test progress.

- ▶ The engine

This component runs on a driving system and executes the script to drive the workload. In our implementation of WebSphere Studio Workload Simulator, we ran the engine in z/OS UNIX System Services.

When running WSWS, the controller communicates with the engine (the driving system) that is actually running the tests. The controller presents a window that allows you to monitor the test progress; refer to Figure 3-6 on page 99.

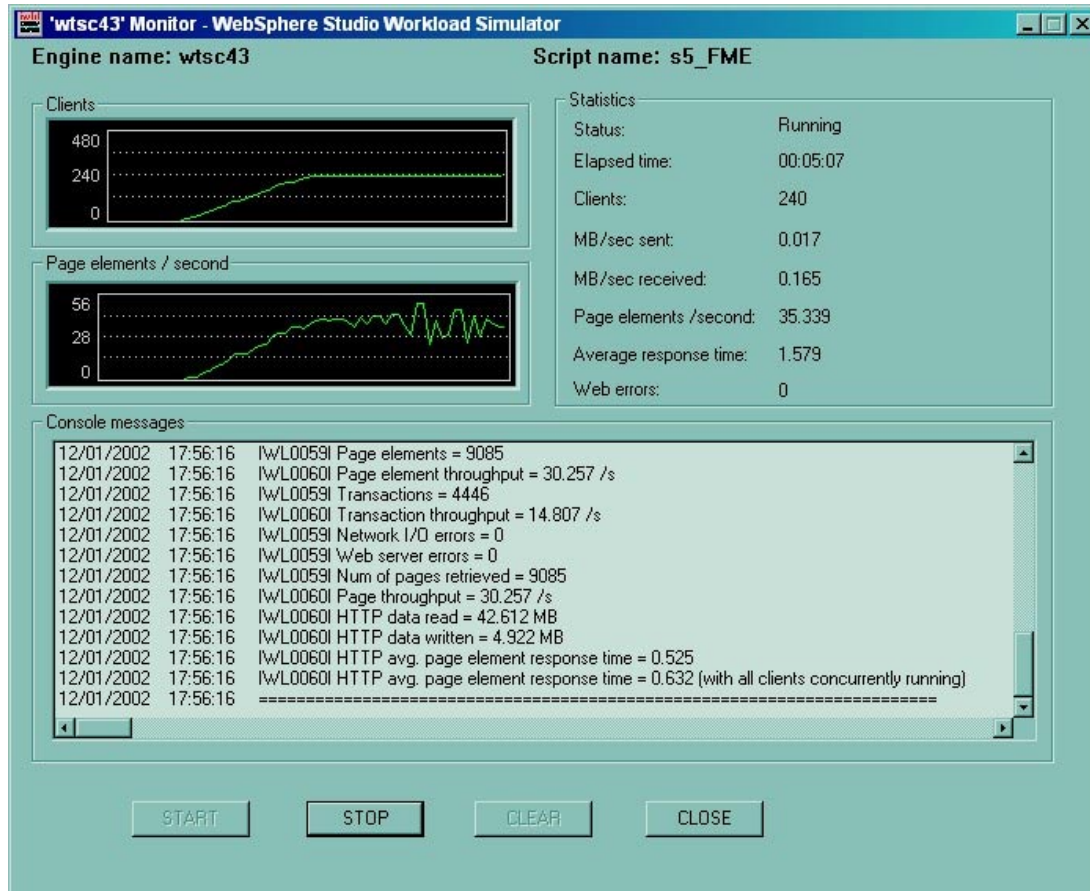


Figure 3-6 WebSphere Studio Workload Simulator

The monitor screen contains a section called “Page elements/second”. This graph shows the number of Web pages per second being returned to WebSphere Studio Workload Simulator over time.

3.2 Examples of performance problems

To demonstrate what can be achieved with specialized WebSphere monitoring tools, we needed some performance problems that might typically be encountered in live customer situations. We sought advice from those with experience in solving real production problems: developers, service personnel, and tool vendors.

From this list of “Frequently Asked Performance Questions”, we derived a set of test cases to illustrate the use of monitoring products.

The examples

In this section we give an overview of each performance problem retained.

Example 1: Identify a DB2 delay in an application path

Problem: A particular transaction normally runs fine, but periodically it shows very poor response time and high system overhead. Show how to identify the specific case or cases where the delay occurs.

The transaction retrieves data from DB2 based on a customer name or account number. For most customers, there is only a small amount of data returned. However, for a small number of very large customers, a huge amount of DB2 data must be retrieved, thus causing big delays.

Example 2: Not used

Problem: Originally based on identifying a JDBC application path, this example was never developed. It somewhat duplicated example 10, and the added value did not justify the cost of development. The example was dropped.

Example 3: Detect a memory leak

Problem: Show how to detect a memory leak. If the problem is allowed to continue, eventually performance degrades and JVM terminates with a Java “out of memory” error.

Example 4: Identify a CICS TS response time problem

Problem: Show how to identify a response time issue caused within CICS TS. Show how one can determine whether the problem is in WebSphere or the CICS Transaction Server.

Example 5: Not used

Problem: Originally based on a JDBC resource shortage, this example was dropped. It is no longer a quantifiable performance issue since WebSphere now dynamically allocates the resources. No measurable performance penalty could be detected on the zSeries model used for testing.

Example 6: Isolate a DB2 problem

Problem: Show how to identify a response time problem caused by accessing a DB2 database.

Example 7: Transaction hang or time-out

Problem: Isolate a transaction that hangs and eventually times out.

All users are complaining about poor response time and operations needs to determine that the application is hung waiting for a response from an external resource (for example, MQ).

Example 8: Static pages serving

Problem: Identify that a large proportion of requests is for static contents, HTML and GIF files rather than JSPs and servlets.

The Edge Server is not caching static files. If the usage is low, or if this can be identified as a normal situation, then the installation may decide to leave well enough alone. Otherwise, the WebSphere Edge Server servers could be configured as caching proxies.

Example 9: Increased WebSphere activity

Problem: Operations detects a large increase in WebSphere activity and overhead. Determine where this additional activity is occurring and decide if this is a problem.

Example 10: Identify a method called with high frequency

Problem: In most cases, a transaction completes quickly, but periodically there is a case where the transaction response time is slow and its CPU overhead is extremely high.

3.3 Performance monitoring tools

The list of situations that we considered is shown in Table 3-4.

Table 3-4 Examples used

Example #	Definition	Notes
1	Identify a path within the application	
2		Dropped
3	Detect a memory leak	
4	CICS TS response time	
5		Dropped
6	Isolate a DB2 problem	
7	Transaction hang or time-out	
8	Static pages serving	

Example #	Definition	Notes
9	Increased activity under WebSphere	
10	Identify a method called with high frequency	
11	Mixed - based on examples 1, 3, 4, 7	

The sequence number assigned has no meaning other than identification. It does not imply any relative priority or logical sequence amongst the different examples.

Using this set of examples, we show in Part 2, “WebSphere performance tools” on page 107, how monitoring tools can help you diagnose the problem and therefore improve the service you provide.

We must emphasize here that using additional performance monitoring tools is not a “magic” solution to every problem. The tools provide extra information, but you need to know how to interpret that information in the context of your own installation. Unless you have that experience, we recommend the services of consultants who can transfer their skills to you while advising on the best methods of setting up performance monitoring.

Introscope

Introscope is a system management application created to help you manage Java application performance. Introscope’s minimal performance impact allows you to monitor and manage your applications’ performance in live production environments.

Chapter 4, “Introscope” on page 109 briefly describes Introscope and explains how it can be used with the above examples. For more information on Introscope, see:

<http://www.wilytech.com/solutions/index.html>

PathWAI

PathWAI solutions is a suite of products that monitor the availability and performance of the systems from one or several designated workstations. It provides many useful reports that you can use to track trends and understand and troubleshoot performance problems.

Chapter 5, “PathWAI solutions for WebSphere” on page 157, describes PathWAI solutions and explains how the products can be used with the examples defined above. For more information on PathWAI solutions, see:

<http://www.candle.com/websphere>

IBM WebSphere Studio Application Monitor

WebSphere Studio Application Monitor for z/OS (WSAM) enables developers and quality assurance and data center personnel to analyze the behavior of applications and take corrective actions to resolve problems. It provides troubleshooting, performance monitoring, and performance analysis functions for large-scale J2EE applications running in development and production environments on the WebSphere for z/OS platform.

Chapter 6, “WebSphere Studio Application Monitor” on page 225 briefly describes WSAM Application Monitor and explains how it can be used with the examples defined above. For more information on WSAM Application Monitor, see:

<http://www-3.ibm.com/software/awdtools/studioapplicationmonitor/>

IBM Tivoli Monitoring for WebSphere Application Server

IBM Tivoli Monitoring for WebSphere Application Server on z/OS is a solution that helps ensure the optimal performance and availability of WebSphere application servers.

Although a chapter on this topic was planned for this redbook, general availability of the product did not match the schedule of our project. Check our Web site for a redbook on IBM Tivoli Monitoring for WebSphere Application Server on z/OS coming later this year:

<http://www.redbooks.ibm.com/>

For up-to-date information on IBM Tivoli Monitoring products, check the Tivoli Developer Domain Web site at:

<http://www-106.ibm.com/developerworks/tivoli/>

Ultimately, this tool increases the effectiveness of an IT organization and provides optimal performance and availability of critical Web infrastructure by:

- ▶ Providing a single point of control to enable IT organizations to understand the health of the key elements of a Web-based environment
- ▶ Letting administrators quickly identify problems, alert appropriate personnel, and offer a means for automated problem correction
- ▶ Providing a real-time view of performance health
- ▶ Feeding a common data warehouse for historical reporting and analysis

Obviously, WebSphere application servers are the key infrastructure pieces of an e-enabled environment. However, IBM Tivoli provides Web infrastructure management capabilities that extend throughout the entire enterprise. A Web infrastructure includes not only instances of WebSphere Application Server, but

also the Web servers that front-end your Web application server, databases on the back-end, message queuing applications, ERP and CRM software, and so on. To enable true management of the entire Web infrastructure, IBM Tivoli provides a single monitoring technology that integrates the management of all these disparate applications.

IBM Tivoli is also focused on driving additional value from your IT investments beyond simply managing and monitoring the various software, hardware, and network configurations that are deployed. This means providing the ability to correlate events and alerts coming from the resources in your IT environment. It also means using the data collected by your monitoring tools to provide feedback and insights into how your IT systems are running and how they can be used to improve your business processes.

Figure 3-7 illustrates the total value proposition of an IBM Tivoli monitoring solution.

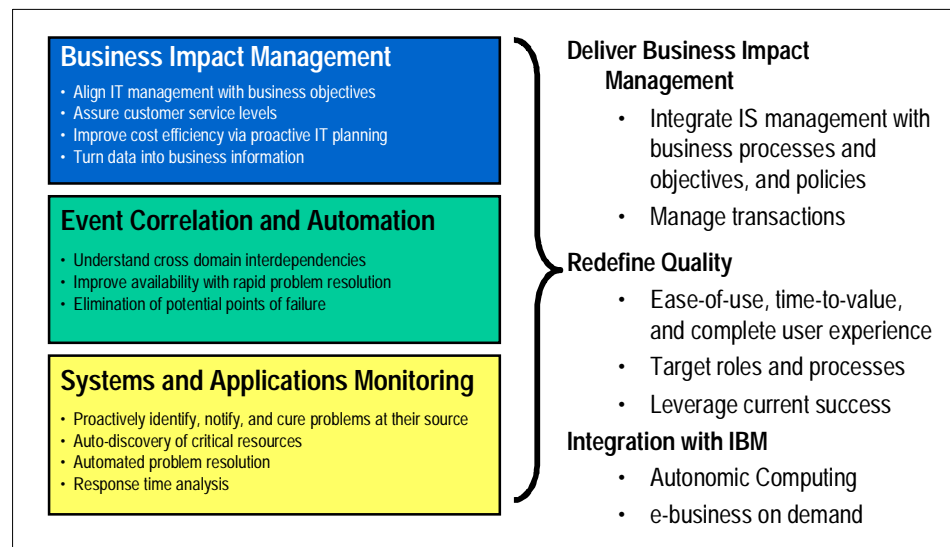


Figure 3-7 IBM Tivoli monitoring solution

Specific to WebSphere Application Server on z/OS, IBM Tivoli Monitoring for WebSphere Application Server on z/OS monitors and manages the critical components of each WebSphere Application Server instance. IBM Tivoli Monitoring for WebSphere Application Server on z/OS deploys Resource Models to proactively monitor an IT environment. Resource Models are combinations of performance metrics that identify specific problem signatures; they also provide alerting capabilities and automated task responses that can solve the problem or prevent it from occurring. Resource Models allow for flexible thresholding, so the

solution can be tailored to meet the unique needs of any environment in which it is deployed. Resource Models also do *persistence checking*, that is, ensuring that problems are chronic and avoiding the unnecessary deployment of IT staff to investigate non-persistent system spikes or irregularities. The Resource Models that ship with IBM Tivoli Monitoring for WebSphere Application Server on z/OS focus specifically on:

- ▶ EJBs
 - Response time
 - Requests per minute
 - Average concurrent
 - Percent discards
- ▶ Web applications
 - Response time
 - Requests per minute
 - Concurrent requests
 - Errors per cycle
- ▶ Transactions
 - Response time
 - Requests per minute
- ▶ HTTP Sessions
 - Active sessions
- ▶ ORB Thread pool
 - Active threads
- ▶ DB2 Connections
 - Wait time
 - Faults
- ▶ JVM
 - Memory used
- ▶ CPU Utilization
 - Per Web app
 - Per EJB

In short, IBM Tivoli Monitoring for WebSphere Application Server on z/OS provides comprehensive management capabilities for your WebSphere Application Server-based e-business environment, by:

- ▶ Ensuring the performance and availability of your application environment including infrastructure components
- ▶ Proactively monitoring critical components of your e-business application based on application best practices
- ▶ Minimizing the risk of outages by conducting thorough root-cause analysis
- ▶ Reducing support and maintenance costs through standardized common administrative tasks

- ▶ Integration with an enterprise-wide data repository for historical reporting and business impact analysis
- ▶ Visualization of the business impact on e-business applications in the context of the corporate IT landscape



Part 2

WebSphere performance tools



Introscope

Wily Technology's Introscope is a comprehensive Web Application Management solution for managing complex Java/J2EE applications on WebSphere for OS/390 and z/OS in live production environments, and for diagnosing a variety of performance problems in production. In addition, Introscope also supports AIX, AS/400®, HP-UX, Solaris, Linux/z, Linux, and Windows.

4.1 Introscope

Wily Introscope monitors applications with the *Whole Application View™* from two perspectives: The application's and the infrastructure's.

From the application's perspective, Introscope reports on:

- WebSphere server regions
 - Web container service requests
 - HTTP sessions
 - JDBC connection pool
- Java/J2EE application
 - Application components including servlets, JSPs, EJBs, and any custom classes and methods
 - J2EE components such as JDBC, JMS, JTA, JNDI, JCA, RMI, XML, and more.
- JDBC driver activity down to the SQL statement level
- Connectors to back-end transaction systems
 - CICS
 - MQSeries
 - IMS
- JVM
 - Memory
 - File I/O
 - Socket I/O

From the infrastructure's perspective, Introscope reports on:

- Web server (Apache, IIS, and more)
 - Number of errors
 - Throughput
- OS/390 and z/OS
 - CPU dispatch time, wait time, process
 - Major subsystems such as DB2, MQ Series, and CICS

4.1.1 Introscope major components

Introscope has three major components: Agents, the Enterprise Manager, and the Workstation.

Agents

Introscope agents collect performance metrics from the various components of the running Java/J2EE applications, WebSphere server regions, and the surrounding computing environment. The various agents collect their performance metrics and report it to the Enterprise Manager.

Enterprise Manager

The Introscope Enterprise Manager coordinates performance metric collection, historical data reporting, alerting, and presentation across all Introscope Agents. The Enterprise Manager acts as the integration point with larger systems monitoring frameworks (for example, Tivoli, OpenView, Unicenter, etc.).

Workstation

The Introscope Workstation is a customizable user interface that visualizes a WebSphere application's performance and its dependent resources. Through the Workstation, users can set alerts on individual metrics or logical metric groups, view performance metrics, and customize views to represent the particular WebSphere environment being monitored.

Figure 4-1 on page 112 illustrates various Java and Environment Performance Agents reporting metrics to the Introscope Enterprise Manager. The Enterprise Manager is integrated with enterprise-wide systems frameworks which receive Introscope alerts and other performance metrics.

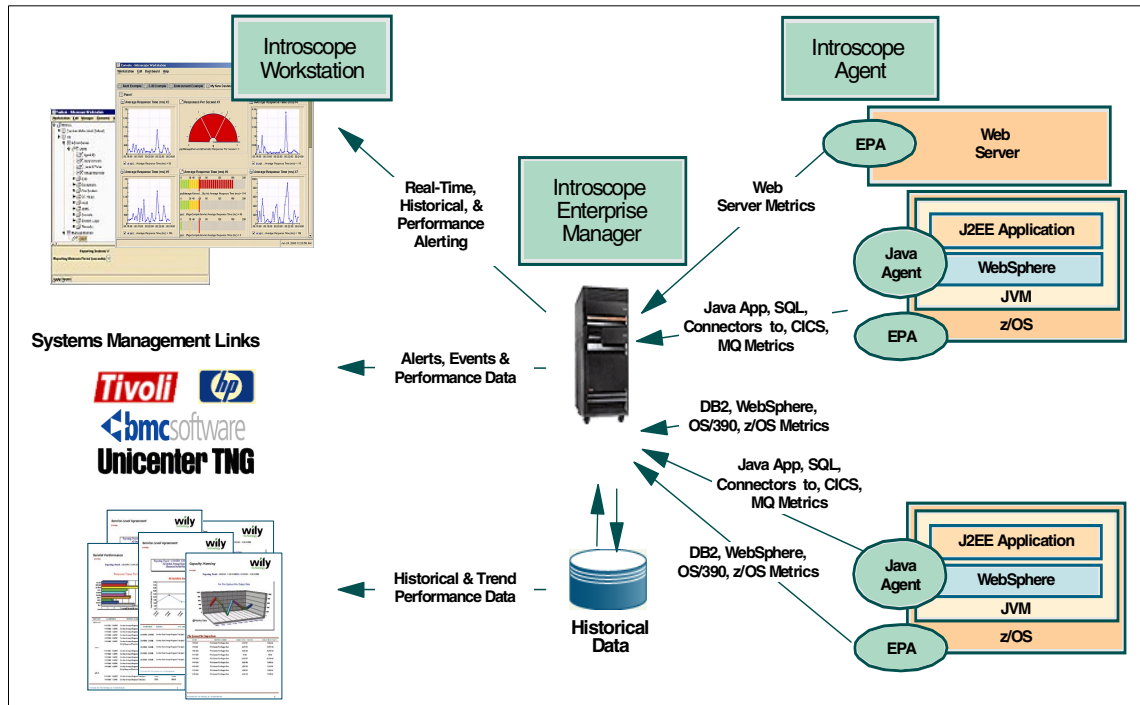


Figure 4-1 Introscope components within existing monitoring frameworks

4.1.2 Monitoring WebSphere on z/OS

Introscope uses two types of agents to collect performance metrics for monitoring the whole application: the Java Agent and the Environment Performance Agent (EPA).

Java Agent

The Introscope Java Agent collects performance metrics on the Java application, WebSphere resources, JDBC, JCA and all other connectors to subsystems such as CICS and MQ Series without requiring developers to write any additional code. See Figure 4-2 on page 113.

The Java Agent monitors Java code running within WebSphere's server regions. In fact, the Java Agent itself runs within WebSphere and does not maintain an address space of its own. The Agent can monitor any Java code, whether or not the code is J2EE compliant, even without the source code. The ability to monitor without the Java source is helpful when monitoring third-party toolkits such as JCA connectors or JDBC drivers.

During its installation and configuration process, the Java Agent can be tailored to monitor key performance metrics of the particular WebSphere application. Typically, this means monitoring the following:

- ▶ The overall response time of each HTTP request from the time the request is dispatched by the WebSphere Request Dispatcher (see Figure 1-1 on page 11 and Figure 1-2 on page 12) to the end of WebSphere's processing of the request
- ▶ The response time and throughput of each J2EE component, for example servlets and EJBs
- ▶ The response time and throughput of each non-J2EE component, for example Apache Struts, custom JSP tag libraries, etc.
- ▶ The response time of all subsystems such as DB2, CICS, MQ Series, IMS, etc.
- ▶ The state of various logical WebSphere resources such as HTTP sessions, JDBC, and other connections
- ▶ The state of various logical application resources such as shared data caches, custom pools, etc.
- ▶ File and socket I/O
- ▶ Application memory for memory leaks
- ▶ The existence of error conditions within the application, leading to poor user experience

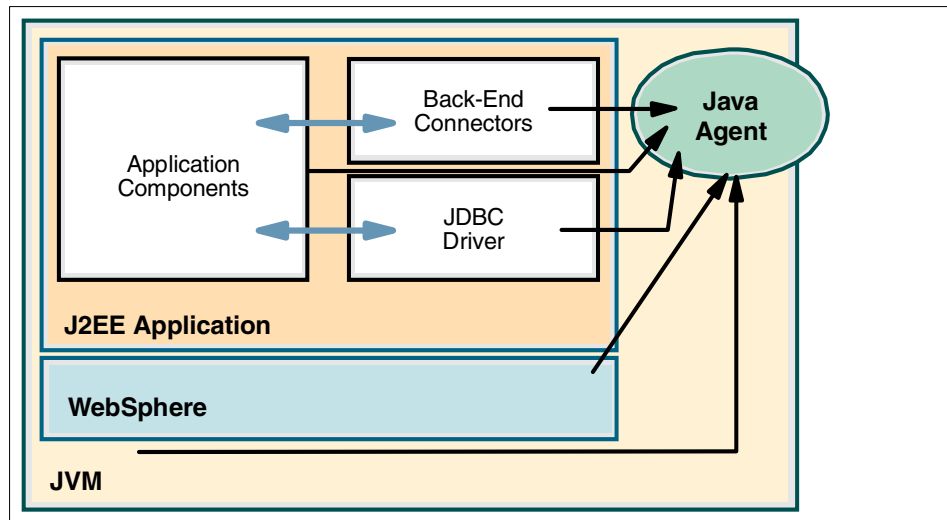


Figure 4-2 Introscope monitors application components and logical and physical resources

In its normal monitoring mode, the Java Agent collects raw performance data for a monitored component or resource over a 15-second interval, aggregates the data over that interval, and reports the aggregated performance metrics to the Enterprise Manager. The Java Agent gathers metrics from every WebSphere request and response; it does not rely on sampling.

In addition to its normal mode, the Java agent can run in *transaction trace* mode. In transaction trace mode, the Java Agent records specific response times of components used by specific user transactions. This is particularly useful for identifying performance bottlenecks in individual user transactions. While providing a great deal more information, transaction trace mode incurs only slightly more overhead than the Java Agent's normal monitoring mode. Switching between normal and transaction trace modes is controlled dynamically from the Introscope Workstation and does not require WebSphere to be restarted. While in transaction trace mode, the Java Agent gathers all the same data as it does in normal mode.

In both modes, the Java Agent associates each component with the components on which it depends. Using this facility, called *Blame Technology*, Introscope can help the user quickly identify which component, among many used in a transaction, is incurring the greatest response times. In normal mode, the Java Agent shows how much of each component's response time is due to other components on average over each 15-second interval. In transaction trace mode, the Java Agent reports how the response times of individual components are incurred by one another within individual transactions.

The Java Agent works with an Introscope component called ProbeBuilder. The ProbeBuilder is tightly integrated into WebSphere class loaders. As an application's code is loaded into WebSphere, the WebSphere class loader sends the code to the ProbeBuilder for *instrumentation*. Instrumentation refers to the process in which the ProbeBuilder inserts monitoring tracers into the code at predefined points described in directive files. These instrumentation points are determined through directive files supplied by Wily Technology. Optionally, users can define their own instrumentation directives to monitor unique application resources.

The directive files supplied by Wily Technology define instrumentation points to monitor the following:

- ▶ Java RMI
- ▶ JDBC
- ▶ JVM I/O
- ▶ CORBA
- ▶ EJBs (Session, Entity, and Message beans)
- ▶ Servlets, JSPs
- ▶ XML, XSL

- ▶ JTA
- ▶ JMS
- ▶ JNDI
- ▶ JavaMail
- ▶ JCA
- ▶ CICS
- ▶ MQ Series
- ▶ WebSphere resources (JDBC pools, thread pools)

Optionally, users can define their own directives to monitor:

- ▶ Application-specific error conditions
- ▶ Application-specific data caches
- ▶ Application-specific connectors that do not conform to J2EE specifications
- ▶ Any custom code down to the method level

Environment Performance Agent

The Introscope Environment Performance Agent (EPA) allows Introscope to receive performance metrics from the computing environment surrounding WebSphere. The EPA provides a mechanism to launch plug-ins that gather metric data and report it back to the Introscope Enterprise Manager. Once launched by the EPA, the plug-in is responsible for obtaining the raw performance metrics and writing these metrics to the STDOUT channel (STDOUT is analogous to z/OS SYSPRINT). The EPA reads the STDOUT and passes the metrics to the Enterprise Manager. See Figure 4-3.

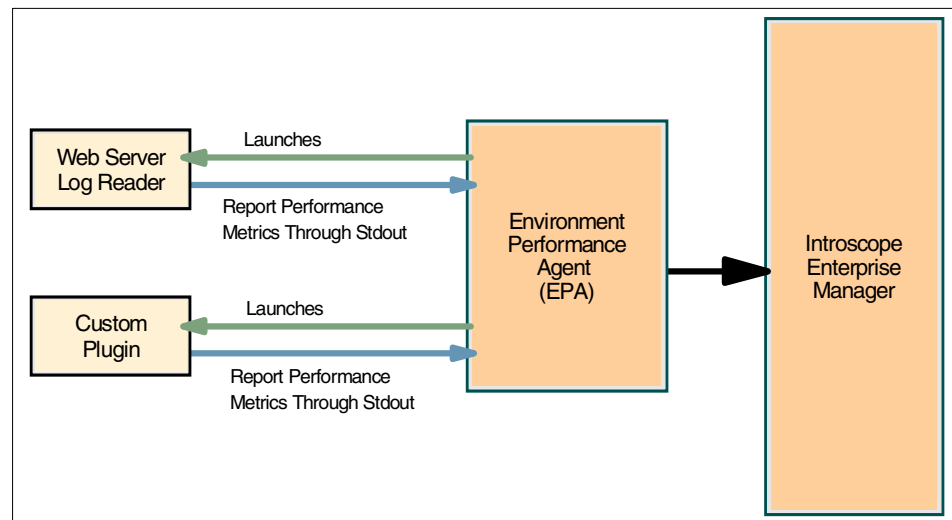


Figure 4-3 The EPA launches plug-ins and receives metrics from the plug-ins via STDOUT

4.1.3 Enterprise Manager

The Enterprise Manager acts as the central collector and coordinator for all Introscope metrics. It collects performance metrics from multiple agents and processes them according to user-defined rules. The Enterprise Manager can generate alerts to a pager, e-mail, or systems management frameworks (for example, Tivoli) if any application component fails to meet application user-defined thresholds. These thresholds can be based on service level agreements, resource utilization, or other meaningful values.

The Enterprise Manager can store metrics to a database for historical analysis and reporting. It also prepares the raw metric data for visualization in the Workstation.

The Enterprise Manager is a Java application and can run on any platform, including OS/390, z/OS, AIX, AS/400, HP-UX, Solaris, Linux/z, Linux, and Windows.

4.1.4 Workstation

The Introscope Workstation consists of three parts: The Explorer, the Console, and the Console Editor.

Explorer

The Introscope Explorer shows a comprehensive list of all application components in a tree format for drill-down problem determination. Figure 4-4 on page 117 shows an example of using the Introscope Explorer to drill down into the components of the application. Notice that the Explorer visually represents component dependencies using the “Called” mechanism.

Console

Introscope’s console graphically represents performance metrics, the larger computing environment, and dependent systems in free-form dashboards. These free-form dashboards allow users to create console views customized to their application environment. Figure 4-5 on page 118 is an example of a dashboard customized for the ITSO sysplex configuration, while Figure 4-6 on page 119 and Figure 4-7 on page 120 illustrate two dashboards as they are customized for the ITSO sample application.

Console Editor

Introscope allows users to completely customize their dashboards in a free-form layout to represent their environment in any manner they choose.

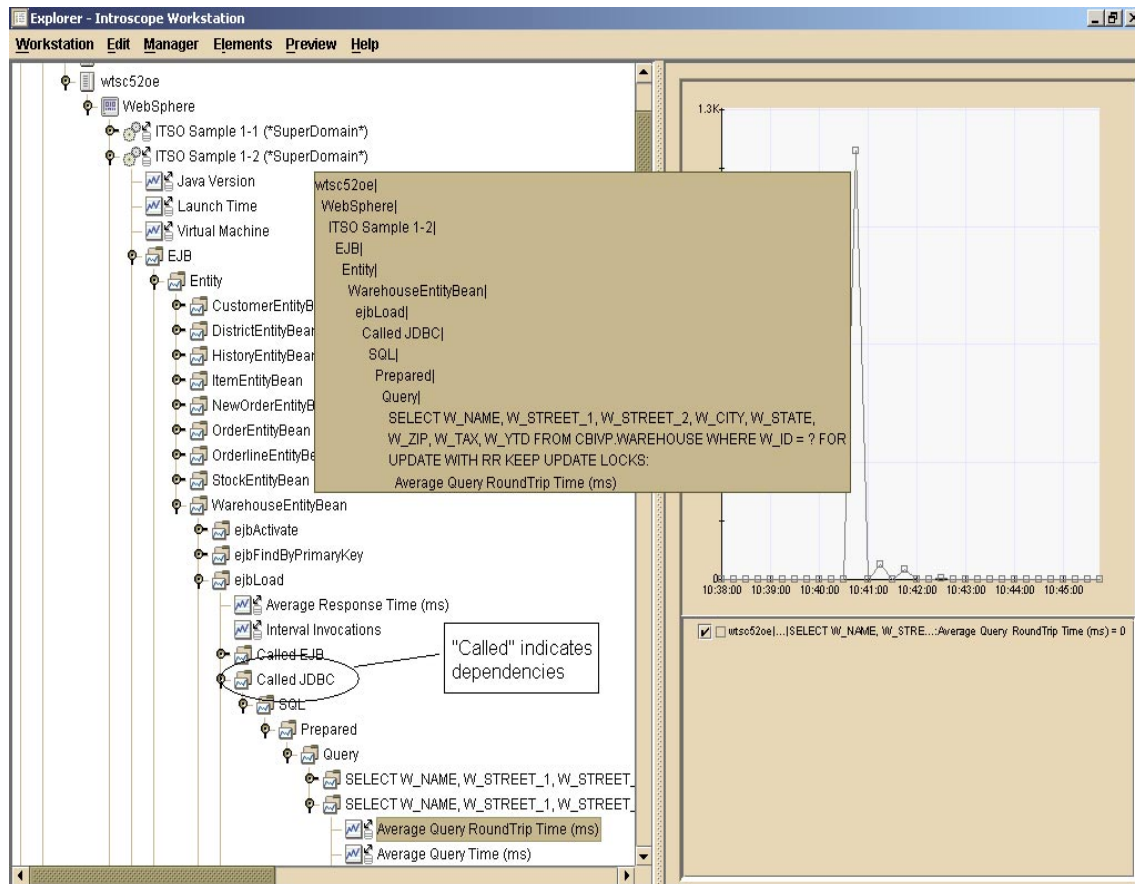


Figure 4-4 The Introscope Explorer is used to drill down into the components of an application

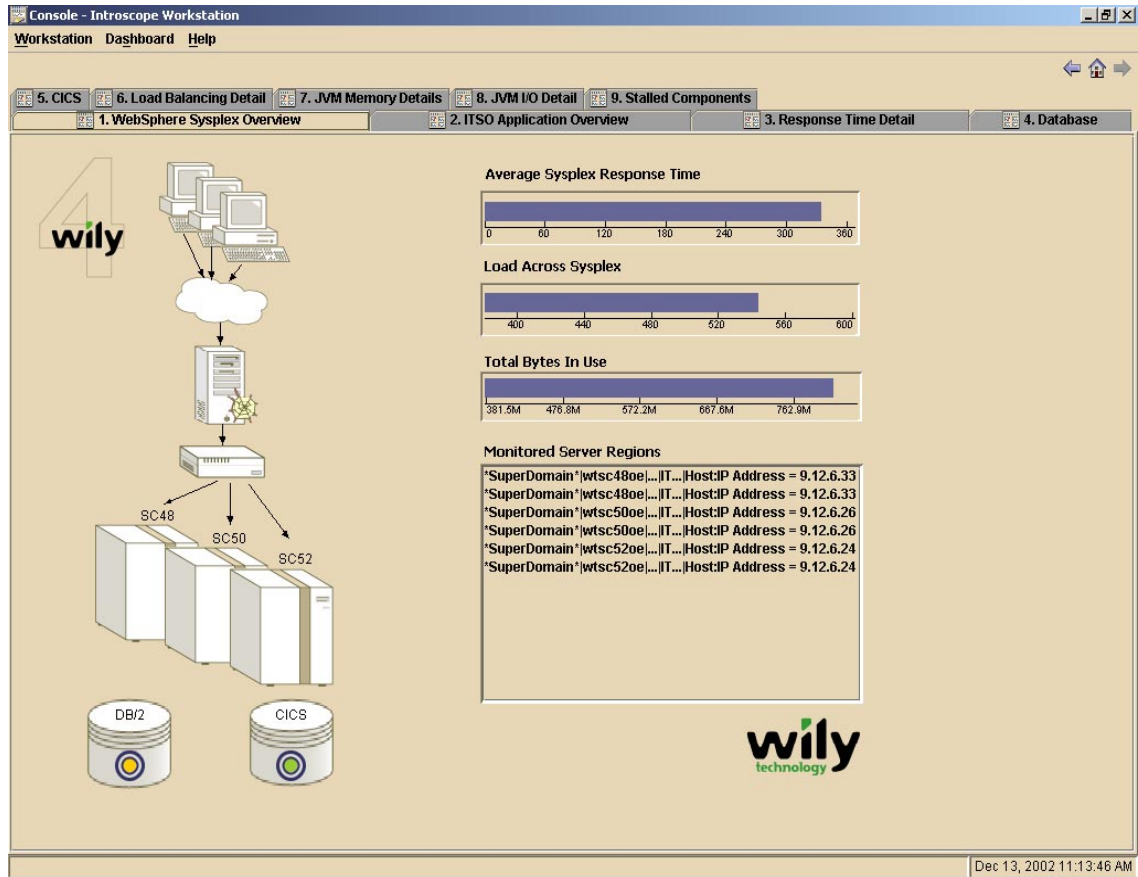


Figure 4-5 Sysplex overview dashboard

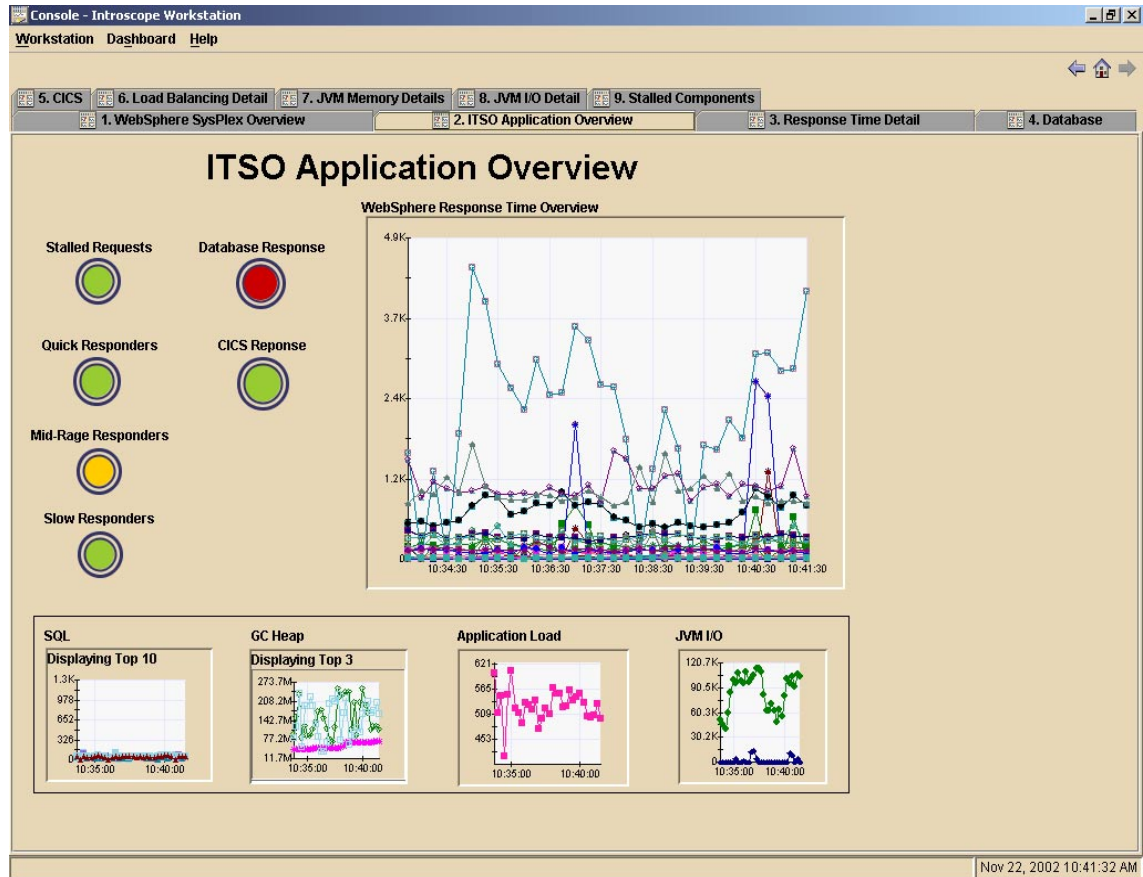


Figure 4-6 Overview dashboard customized for the ITSO application

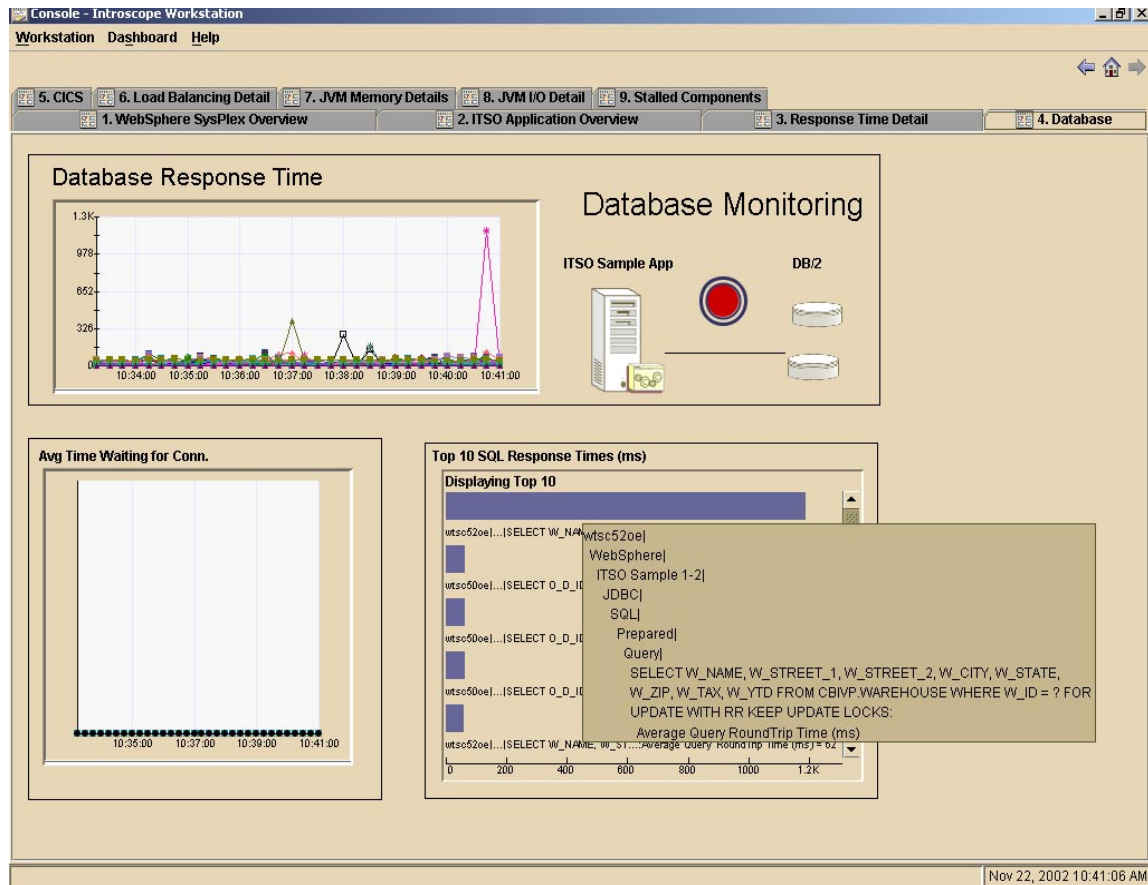


Figure 4-7 Dashboard to monitor the ITSO application's use of DB2

The Introscope Workstation also serves as the launch point for two other Introscope extensions: Transaction Tracer and Leak Hunter.

Transaction Tracer

Transaction Tracer helps isolate performance problems by visualizing the components and their dependencies on individual user transactions to pinpoint bottlenecks. Figure 4-8 on page 121 shows an example of Transaction Tracer in use.

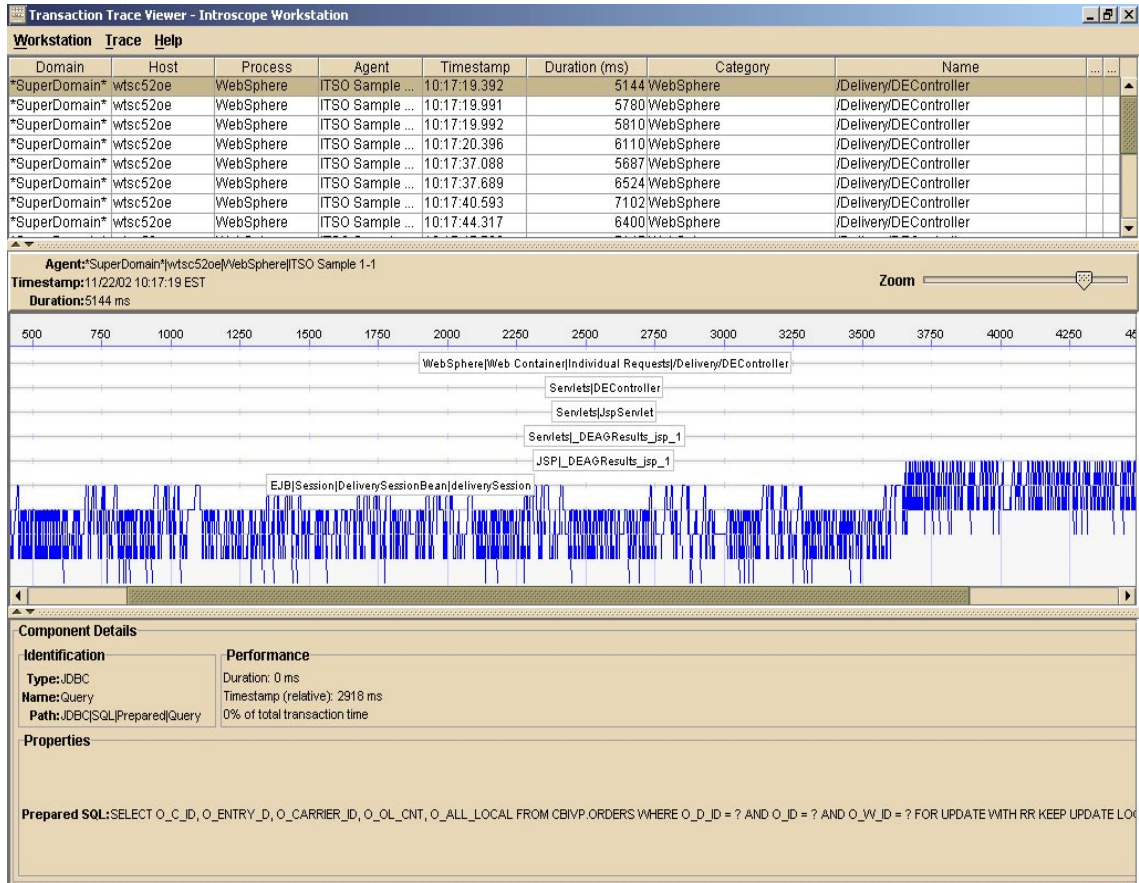


Figure 4-8 Introscope's Transaction Tracer allows the user to quickly visualize individual transactions

Figure 4-9 shows an example of the top portion of the Transaction Tracer window listing all individual transactions that exceed a performance threshold. The user selects a transaction from the list to see a complete breakdown of all its components and their performance characteristics.

Host	Process	Agent	Timestamp	Duration (ms)	Category	Name
wtsc52oe	WebSphere	ITSO Sample ...	10:17:19.392	5144	WebSphere	/Delivery/DEController
wtsc52oe	WebSphere	ITSO Sample ...	10:17:19.991	5780	WebSphere	/Delivery/DEController
wtsc52oe	WebSphere	ITSO Sample ...	10:17:19.992	5810	WebSphere	/Delivery/DEController
wtsc52oe	WebSphere	ITSO Sample ...	10:17:20.396	6110	WebSphere	/Delivery/DEController
wtsc52oe	WebSphere	ITSO Sample ...	10:17:37.088	5687	WebSphere	/Delivery/DEController
wtsc52oe	WebSphere	ITSO Sample ...	10:17:37.689	6524	WebSphere	/Delivery/DEController

Figure 4-9 Transaction Tracer captures a list of all individual transactions that exceed a user-defined response time threshold

Figure 4-10 illustrates the component breakdown of one of the transactions, which is displayed in the middle portion of the window. The time in the transaction is represented horizontally across the window. From top to bottom are the application components that are used within the transaction. This unique visualization allows users to quickly see an entire transaction, its component parts, and the dependencies.

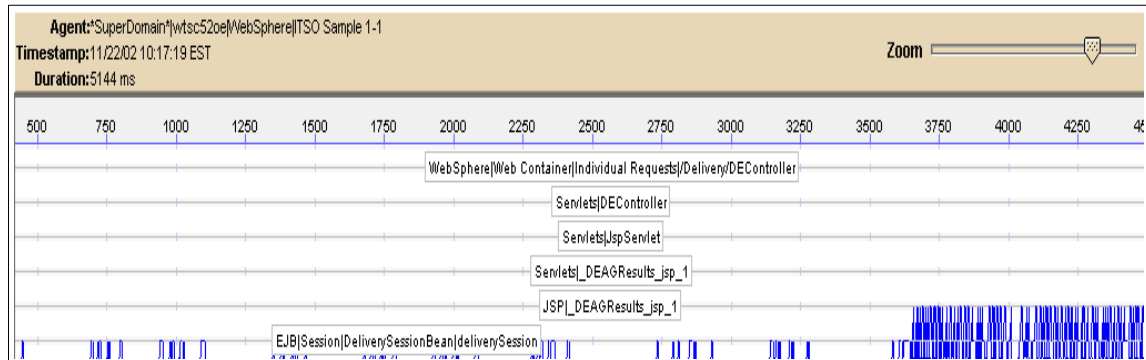


Figure 4-10 Response time is represented horizontally while components used by the transaction are shown vertically

The user can select any one of the components and see its details, as illustrated in Figure 4-11.

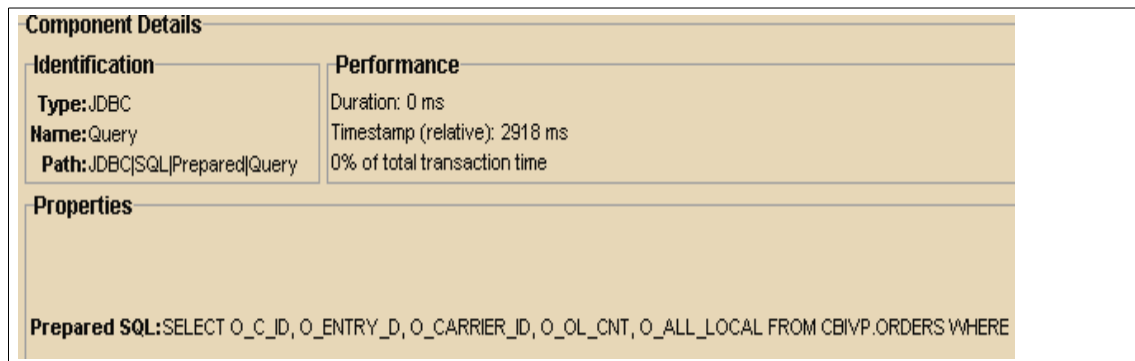


Figure 4-11 Details of the component are shown in the bottom portion of the Transaction Tracer window

Leak Hunter

Leak Hunter can isolate application memory leaks. While the associated overhead is small enough to use in production, most users will choose to enable Leak Hunter only when a leak is suspected. Once a memory leak is detected, Leak Hunter provides information for developers to quickly identify the leaking component.

4.1.5 Introscope performance and monitoring methodology

In general, Introscope approaches performance problems from the standpoint of isolating bottlenecks in the course of executing a transaction. These bottlenecks fall into two broad categories: the time spent waiting on a resource and the time spent using the resource. Introscope monitors individual transaction requests entering WebSphere and tracks the use of both logical and physical resources within each application component. By examining both the time spent waiting on a resource and the time spent using the resource, Introscope helps identify the source of application bottlenecks.

Through the customization process, Introscope is set up to monitor application-specific resources such as data caches, non-J2EE components, and specialized back-end systems. Although the particular metrics for each of these resources will be different, the idea is to provide a measure of each resource's utilization, queue time, and execution time.

In production, Introscope is typically configured with response time alerts for incoming WebSphere requests. In complex applications, some of these requests may naturally have longer response times than others. In these cases, we configure Introscope to group requests into various response time categories where each category has its own set of response time thresholds. This helps prevent Introscope from alerting on conditions that are normal.

In addition to response time alerts for all WebSphere requests, it is important to alert on the response times for each of the subsystems supporting the application (such as DB2, CICS, IMS, etc.).

In practice, however, it is sometimes difficult to identify all of the back-end systems. In these cases, we identify as many subsystems as possible, group the requests into response time categories, and set appropriate alerting thresholds. Should the unidentified subsystems cause the application to respond slowly, monitoring each of the incoming WebSphere requests will indicate a problem. Using Introscope's ability to drill down within the application (with the Explorer) and visualize individual transactions (with Transaction Tracer), the offending subsystem can be identified quickly.

4.1.6 ITSO configuration

Introscope Java Agents were deployed during the test runs at the ITSO. As Figure 4-1 on page 112 shows, running within the address space of each WebSphere server region is an Introscope Java Agent monitoring the WebSphere application. Various Introscope Workstations were configured on a variety of different machines for easy monitoring.

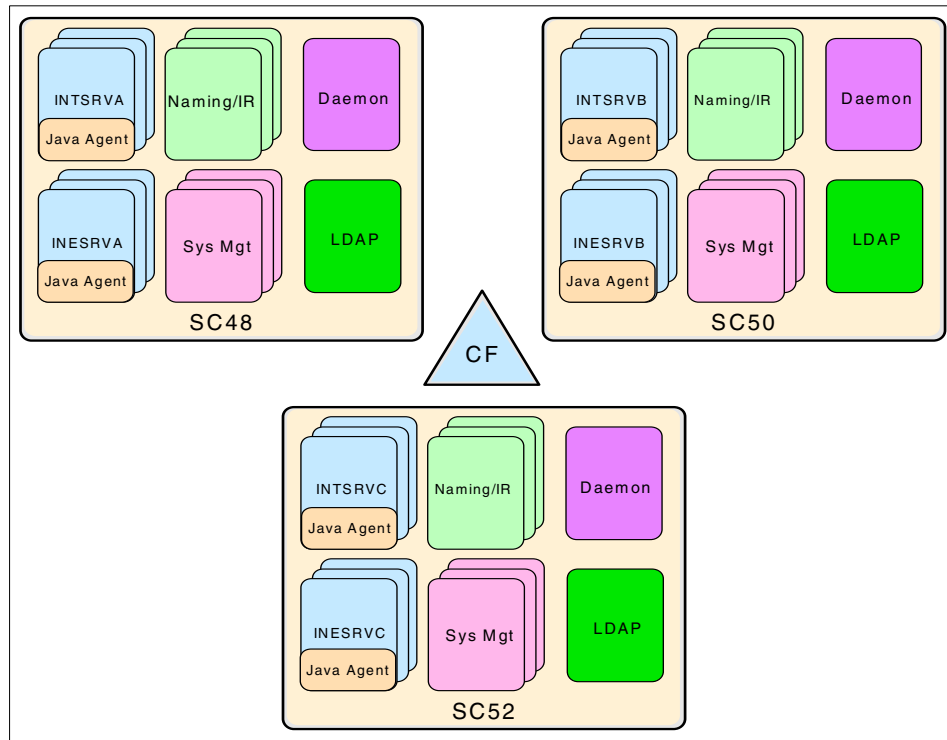


Figure 4-12 Introscope Java Agent configuration at the ITSO

Using the directive files supplied by Wily Technology, the Java Agents reported the following metrics:

- ▶ The response time for each URI entering the WebSphere server region from the moment it is dispatch by the request dispatcher to the time that the application code is finished handling the request
- ▶ The response times and throughput of each application servlet, JSP, and EJB
- ▶ Any in-flight WebSphere transactions that have been in the server region for more than 30 seconds
- ▶ Any application component that received a request but has not responded within 30 seconds
- ▶ The response times and throughput of individual SQL statements
- ▶ The elapsed time required to process result sets from each SQL SELECT statement
- ▶ The elapsed time to establish a connection to DB2
- ▶ The response times and throughput of CICS transactions

- ▶ The response time to get and put messages on MQ Series queues
- ▶ The number of in-use JDBC connections
- ▶ The application's use of memory including memory leak detection

In addition to the metrics from the Wily Technology directives, the Java Agent was configured to report:

- ▶ The rate of exceptions (errors) processed by the ITSO sample application

All of these metrics are collected by the Java Agent monitoring the WebSphere server regions and the application code. The Java Agent does not rely on data from SMF 120 and does not require the user to turn on this record type.

In addition to raw metrics, Introscope was configured with a number of alerts as they would be in a typical production deployment:

- ▶ Any DB2 response that exceeds 500 milliseconds.
- ▶ WebSphere HTTP requests are categorized into quick, mid-range, and slow categories based on the normal response times of the request. If the response time for a particular HTTP request is longer than the threshold defined for its category, an alert is triggered. The response time thresholds of each category are:
 - Quick responders must have a response time of under 800 ms.
 - Mid-range responders must have a response time of under 1600 ms.
 - Slow responders must have a response time of under 7000 ms.
- ▶ Any in-flight WebSphere request not responding to a request within 30 seconds.
- ▶ Any application component not responding to a request within 30 seconds.

The first two alerts are *response time alerts* that report the elapsed time to complete a transaction. The last two alerts are *stalled request alerts* and are triggered immediately when a request takes longer than 30 seconds to complete. If one of these long-running transactions completes after thirty seconds, the alert is cleared.

The difference between response time alerts and stalled request alerts is best illustrated with an example. Suppose an HTTP request enters the WebSphere server region and takes 45 seconds before it returns a response. Thirty seconds after the request enters the server region, the Java Agent increases the WebSphere HTTP stalled request counter from 0 to 1, and the Workstation receives a stalled request alert. Fifteen seconds later, when the server region finishes processing the request, the Java Agent decreases the WebSphere

HTTP stalled request counter from 1 to 0, it records a 45-second response time and the stalled request on the Workstation is cleared.

The stalled request alerting logic can be applied to any component in the application. Placing stalled request counters on many different application components allows Introscope users to quickly identify what component is causing a transaction to hang.

The exact values for all of these alert thresholds are set through testing, experience, or service level agreements, and can be configured dynamically within Introscope. In fact, the responder categories themselves are an arbitrary grouping of HTTP requests and not predefined by Introscope.

In a typical production deployment, Introscope alerts would be integrated into a larger enterprise-wide management framework such as Tivoli, CA UniCenter, or HP OpenView.

4.2 Examples

The following scenarios illustrate methodology described in 4.1.5, “Introscope performance and monitoring methodology” on page 123 as applied to specific performance scenarios in production. The intent of each example is to describe how Introscope would help isolate performance problems in a typical production environment.

4.2.1 Example 4: CICS

Introscope has sent an alert indicating that a particular HTTP request in the “quick responders” group is responding slower than the threshold. A quick look at the “quick response” category indicates, in Figure 4-13 on page 127, that there are a few requests responding well above the alerting threshold.

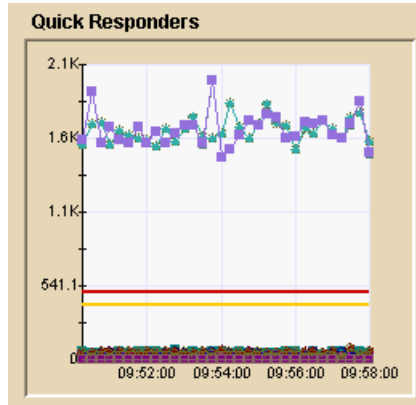


Figure 4-13 Response times of the “quick responders” group. A few HTTP requests across the sysplex are above the threshold.

No easily testable hypotheses come immediately to mind and we need to gather additional data. We dynamically switch the Introscope Java Agents into transaction trace mode and examine the results illustrated in Figure 4-14.

Host	Process	Agent	Timestamp	Duration (ms)	Category	Name
wtsc480e	WebSphere	ITSO Sample 2-2	09:58:58.729	2048	WebSphere	/jms/JMSController
wtsc480e	WebSphere	ITSO Sample 2-2	09:58:58.738	2051	WebSphere	/jms/JMSController
wtsc480e	WebSphere	ITSO Sample 2-2	09:58:58.743	2056	WebSphere	/jms/JMSController
wtsc480e	WebSphere	ITSO Sample 2-1	09:58:57.139	2065	WebSphere	/jms/JMSController
wtsc480e	WebSphere	ITSO Sample 2-1	09:58:57.663	2066	WebSphere	/jms/JMSController
wtsc480e	WebSphere	ITSO Sample 2-2	09:58:49.267	2073	WebSphere	/jms/JMSController
wtsc480e	WebSphere	ITSO Sample 2-2	09:58:47.692	2075	WebSphere	/jms/JMSController
wtsc480e	WebSphere	ITSO Sample 2-2	09:58:59.226	2087	WebSphere	/jms/JMSController

Figure 4-14 All of the slow HTTP requests have the same URI, /jms/JMSController

Selecting one of the HTTP requests, we can see its complete response time breakdown by component in Figure 4-15 on page 128. We see that the overall request took just over 2000 ms and uses several components, including the CTG Client and CTG Servers. These appear to constitute the majority of the response time.

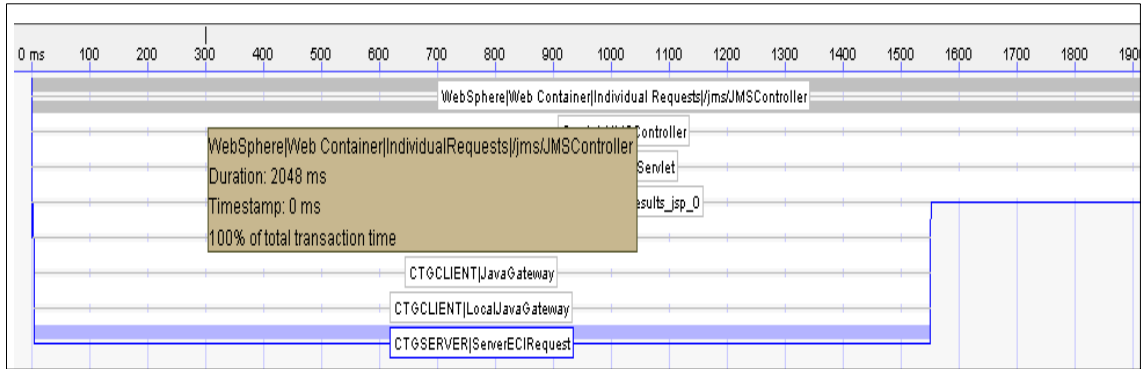


Figure 4-15 The complete response time breakdown of an individual HTTP request by component

In Figure 4-16, by selecting the CTG Server component, we discover that this component contributes 1.5 seconds (over three quarters) of the overall response time. This component is performing the ServerECIRequest function, which executes the CICS transaction.

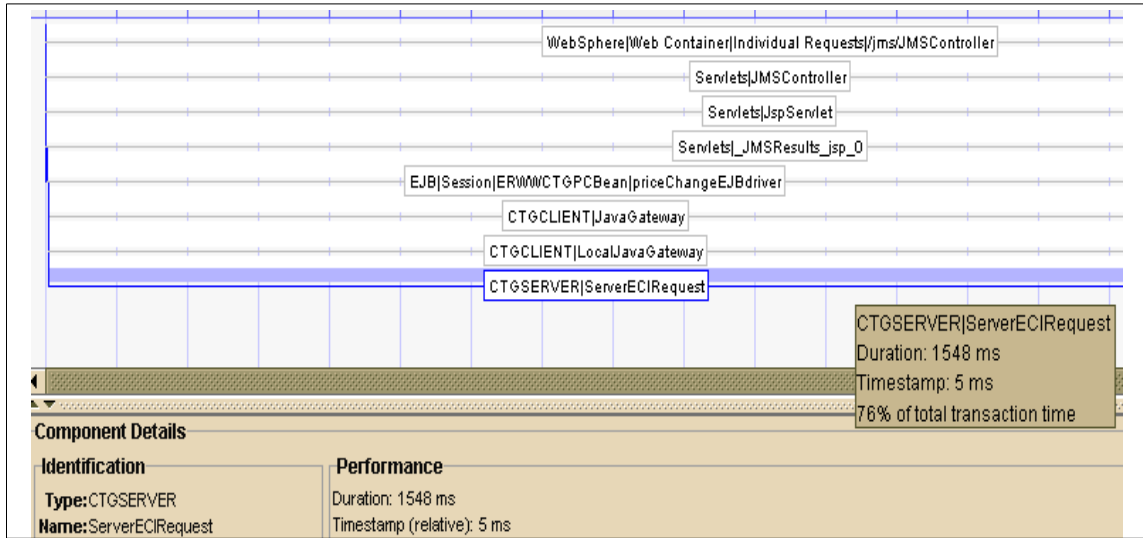


Figure 4-16 The CTG Server executing the CICS transaction constitutes over 75% of the response time

Examining the other transactions listed in Introscope's Transaction Tracer list (again, see Figure 4-14 on page 127), we see almost identical results. Therefore, it is safe to assume that WebSphere's response time problems are due to a slow-responding CICS transaction.

Conclusion

This example illustrates the importance of monitoring back-end connections from the application's point of view. In practice, a CICS region may be shared across many different applications. Knowing transaction response times from CICS point of view may not help isolate the problem due to the sheer volume of data involved. However, monitoring CICS response times from the application's point of view clearly shows the source of the problem.

This example also shows that even though no alerts were set on CICS response times when Introscope was initially configured, we were able to detect a performance problem by monitoring each of the incoming HTTP requests. Then, by isolating the poorly performing transactions using Transaction Tracer, we were able to quickly identify the offending component. Now, armed with the knowledge that CICS can be a major contributor to application response time, we dynamically configure a response time alert on CICS.

4.2.2 Example 6: No DB2 Index

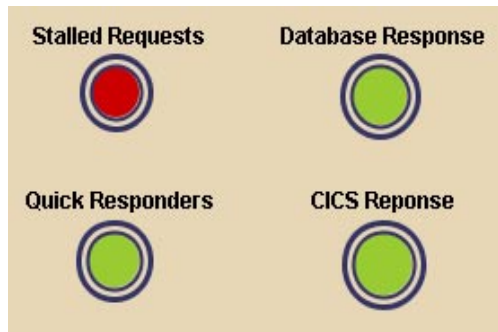


Figure 4-17 Stalled Request Alert has turned red

In Figure 4-17, we see that Introscope has alerted us that at least one in-flight WebSphere request has not responded within 30 seconds. Introscope defines a stall as a Java method that has begun executing code, but has not returned a response within some predefined time-out. In this case, the time-out was configured to be 30 seconds.

Looking at these stalled requests in more detail, Figure 4-18 on page 130 shows that the `ejbFindByPrimaryKey` method of the `CustomerEntityBean` has begun processing a request but has not responded within 30 seconds (hence, its “Stalled Method Count” is one). It is curious that an `ejbFindByPrimaryKey` method should take longer than 30 seconds because, as the name implies, it is a database lookup on an indexed DB2 table.

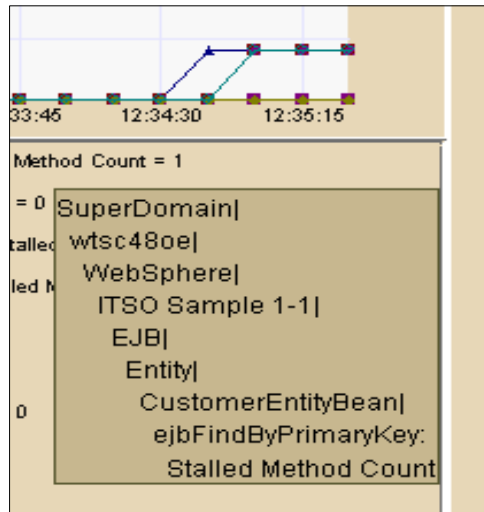


Figure 4-18 The `ejbFindByPrimaryKey` method in the `CustomerEntityBean` has stalled on one request

A couple of hypotheses come to mind:

- ▶ It is possible that DB2 is experiencing problems.
- ▶ The application's connection to DB2 is somehow broken.

In either of these cases, we would expect that all DB2 accesses would somehow display signs of problems. If this were the case, we would expect most response times to be higher than normal. However, Figure 4-19 on page 131 indicates that WebSphere has been responding to HTTP requests in under 40 ms for several minutes.

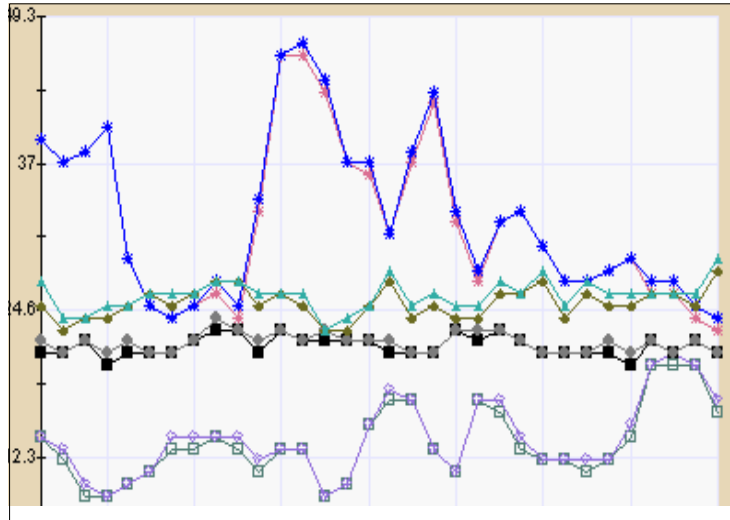


Figure 4-19 WebSphere response times per HTTP request

Taking a closer look at DB2:

- ▶ Figure 4-20 shows that the top four of the worst ten response times over the past several minutes have response times of under 2 ms.
- ▶ Figure 4-21 on page 132 shows that the application is able to obtain a connection to DB2 very quickly.

It appears that our initial hypotheses are incorrect and we must obtain additional information to form new ones.

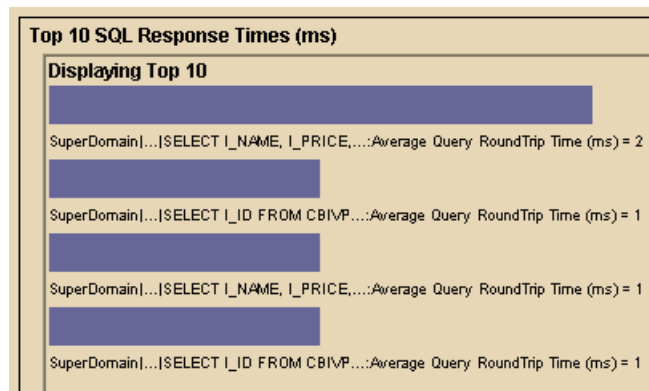


Figure 4-20 Top four of the worst ten SQL response times; all are 2 ms or less

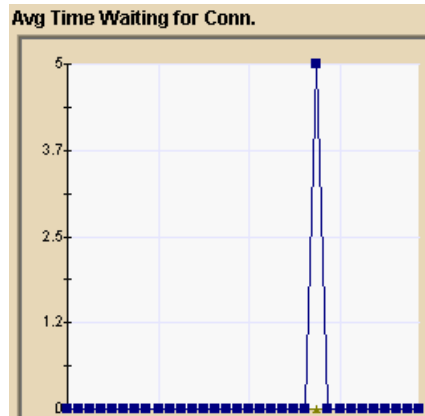


Figure 4-21 Average time the application must wait for a DB2 connection (in ms)

While trying to determine what additional information to gather, we see the alert status change. Figure 4-22 shows the Stalled Request alert has cleared, but now Introscope indicates that database response time is slow. Checking the database response times again in Figure 4-23 on page 133, we see that a particular query, `SELECT ... FROM CBNP.CUSTOMER WHERE C_ID = ? AND C_D_ID = ? AND C_W_ID = ?`, has taken approximately two and a half minutes to complete.



Figure 4-22 The Stalled Request alert has cleared and the Database Response alert turns red

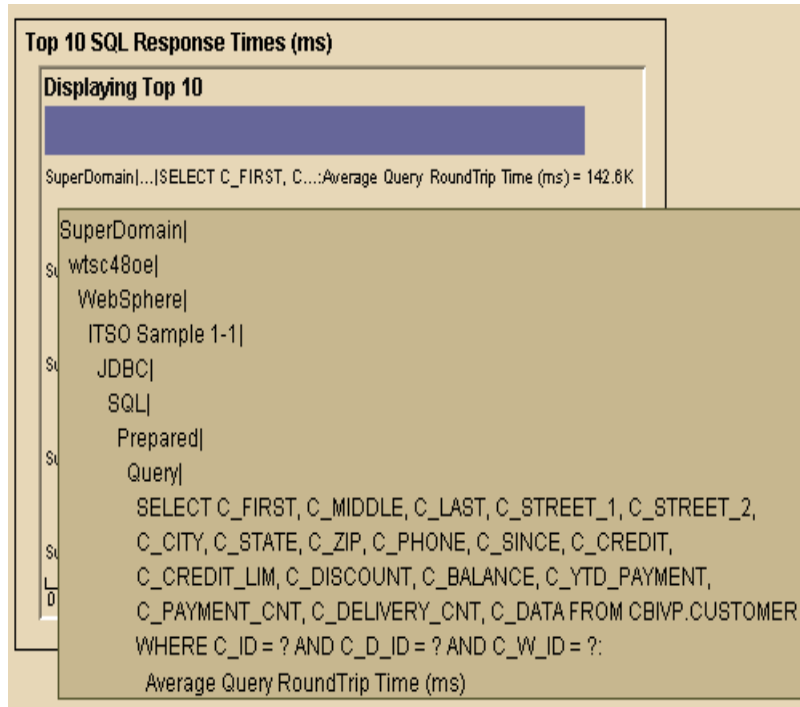


Figure 4-23 This query has taken 142.6 thousand milliseconds to complete

The fact that the response time for this SELECT statement exceeded 30 seconds explains why Introscope first detected the problem as a stalled request. Introscope is configured to alert on any request that has been in the system for longer than 30 seconds, regardless of what its ultimate response time will turn out to be. When the database finally responded to the request two minutes later, Introscope alerted on a slow database response.

Conclusion

It is important to note that, in this example, measuring average database response time would have been inadequate to identify the source of the problem. In fact, it is quite possible that averaging the bad database response times with many good responses would have completely masked the problem. In practice, monitoring back-end responses as granularly as possible (for example by SQL statement, CICS transaction ID, MQ Series queue name) is invaluable in both detecting and diagnosing performance problems.

4.2.3 Example 10: Too Much Logging

Figure 4-24 shows all categories of HTTP requests (quick, mid-range, and slow) are above their respective alerting thresholds. Because the Java Agent captures performance metrics from the time the transaction is handled by the WebSphere request dispatcher until the server region is finished handling the request, we can be confident that the poor response time is caused by an application component or a resource utilized by the server region as opposed to the network or Web server.

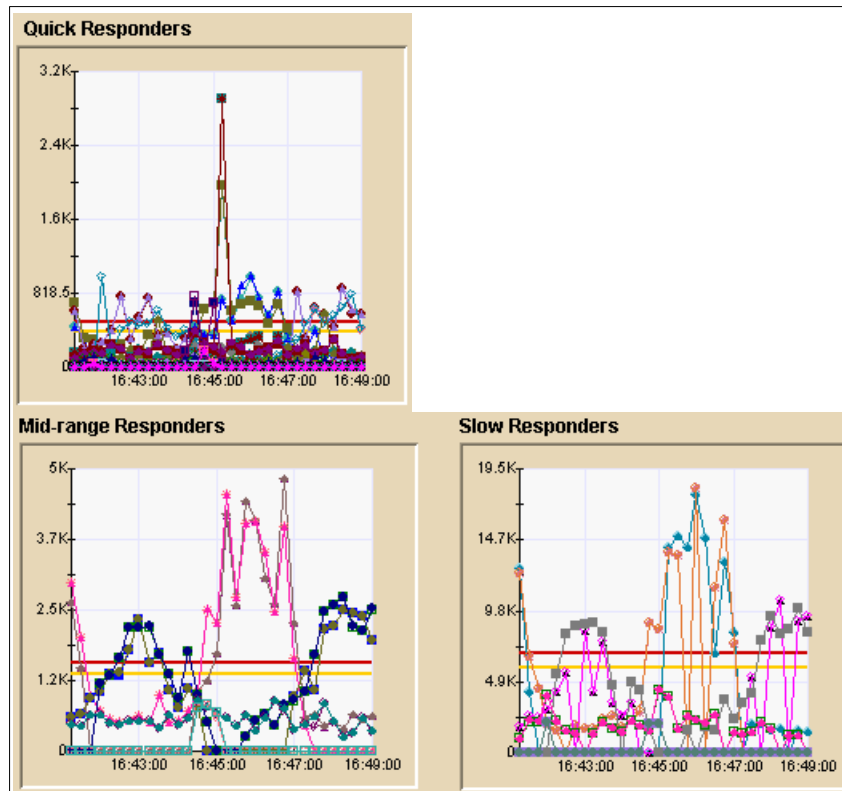


Figure 4-24 Each request response time group is above the alerting threshold

Several hypotheses come immediately to mind:

- ▶ Higher load on the system.
- ▶ An application memory leak.
- ▶ The CPU available to the WebSphere LPAR has changed.
- ▶ A back-end resource common to most HTTP requests is slow.
- ▶ A hardware error is causing a high number of interrupts.

With so many hypotheses to test, we try to gather more information in the hopes of somehow prioritizing our investigation. One of the items that stands out from a quick system overview is Figure 4-25. The server regions' I/O rates are far higher than normal. This server region is generating 84,000 bytes per second of output. Other server regions have a similarly high rate of output.

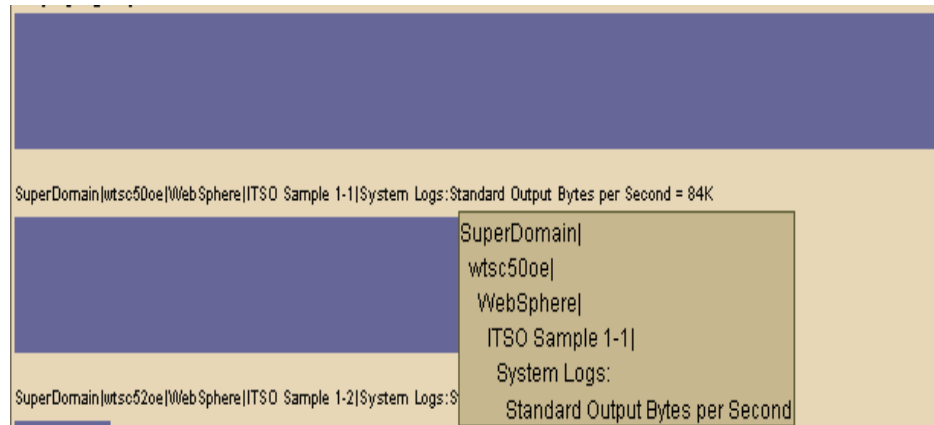


Figure 4-25 Nearly 85 kps of output from multiple server regions

Grouping the rate of output with the application code generating the output, we can see why. Figure 4-26 on page 136 indicates that the debugOut method is the culprit.

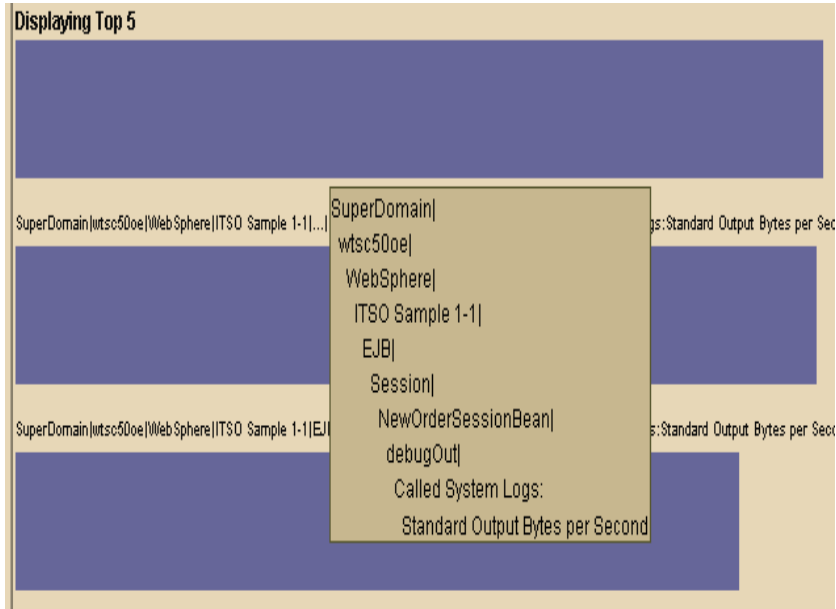


Figure 4-26 The debugOut method tops the list of methods causing system output

The very name, debugOut, indicates that it is likely that a logging configuration parameter is set incorrectly. Upon investigation, the logging level for the application is set to *debug* instead of *production*.

Conclusion

This example illustrates the importance of judging the evidence from the monitoring tools against one's expectations of the system's behavior. Without an expectation that the system should not be performing a lot of I/O, one could easily miss the incorrect log level setting and begin an exercise of investigating several false hypotheses.

4.2.4 Example 3: Memory Leak

Figure 4-27 on page 137 indicates that a WebSphere server region's memory requirements have been growing. All other server regions' memory graphs indicate the same thing. Of course, this does not immediately imply a memory leak in the application, but it does fit the pattern described in Appendix B, "Configuration files" on page 291: the minimum points are ever increasing.

Figure 4-28 on page 137 shows that the overall average response time across the entire sysplex has been steadily growing for several minutes.

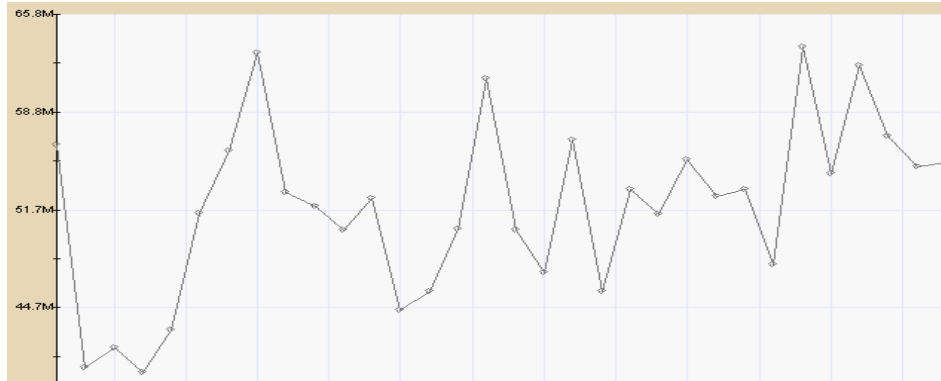


Figure 4-27 Average heap size of this WebSphere server region is growing

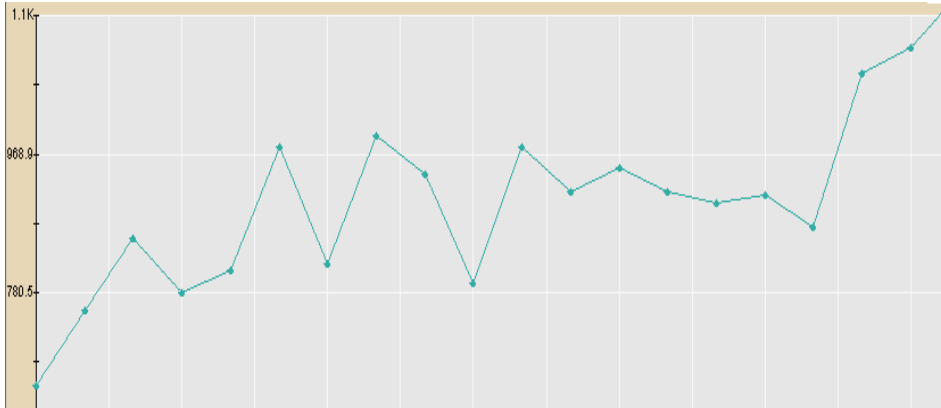


Figure 4-28 Average response time across all WebSphere requests in the sysplex

There are two immediate hypotheses:

- ▶ The load on the system is increasing, causing more memory to be consumed.
- ▶ The application has a memory leak. Under this hypothesis, the increased response times would be due to the Garbage Collector taking more CPU cycles.

Investigating the first hypothesis, we initially expect that additional load would imply more HTTP sessions in use to handle the load. However, Figure 4-29 on page 138 shows no obvious increase in the number of HTTP sessions.

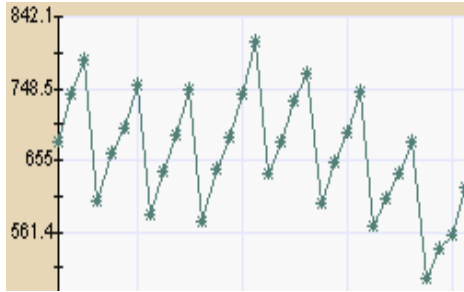


Figure 4-29 HTTP live session count over time

Further, a graph of system load in Figure 4-30 does not indicate any growth in load over the same time period as the server regions' increase in memory. In fact, it appears that the load on the system has been decreasing. It is possible that the application has a memory leak.

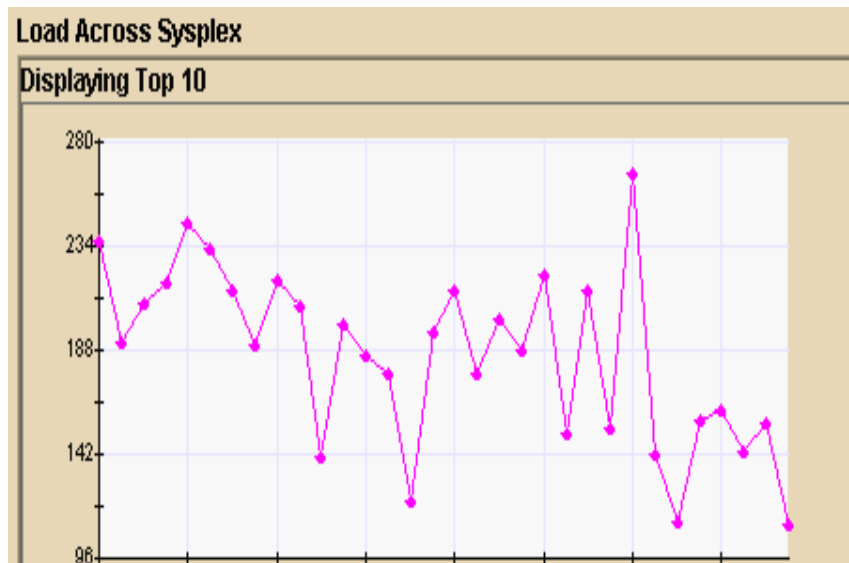


Figure 4-30 Application load has been decreasing over time

After enabling Leak Hunter and restarting the server region JVM, Introscope now tracks WebSphere's server regions' memory allocations. After several minutes of load, Figure 4-31 on page 139 shows that Leak Hunter has detected a potential memory leak.

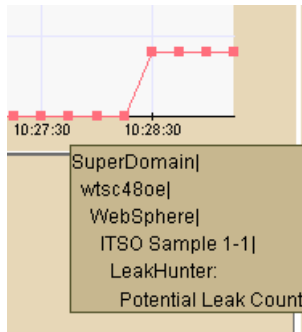


Figure 4-31 Leak Hunter detects the source of the memory leak

Figure 4-32 shows a Java stack trace indicating which application is causing the leak, and where in that application the leaking component is. With this knowledge, the problematic application can be isolated in its own server region and managed independently as described in “Managing memory leaks in production” on page 285. This information can also be given to the application’s developers to help resolve the problem.

```

SuperDomain|wtsc480e|...|<clinit>-1991#1:Allocation Stack Trace =
at weberwwno.NOController.<clinit>(NOController.java)
at java.lang.Class.newInstance0(Native Method)
at java.lang.Class.newInstance(Class.java:262)
at java.beans.Beans.instantiate(Beans.java:233)
at java.beans.Beans.instantiate(Beans.java:77)
at com.ibm.servlet.engine.webapp.WebAppServletManager.loadSer
at com.ibm.servlet.engine.webapp.WebApp.loadServletUnderRemo
at com.ibm.ws390.wc.webapp.RemoteWebAppImpl.driveLoadServ
at com.ibm.ws390.wc.container.RemoteWebAppBean.driveLoadSe
at com.ibm.ws390.wc.container.EJSRemoteStatelessRemoteWeb
at com.ibm.ws390.wc.container.RemoteWebContainerBean.driveL
at com.ibm.ws390.wc.container.EJSRemoteStatelessRemoteWeb

```

Figure 4-32 Leak Hunter information can be given to developers to fix the problem

Conclusion

Memory leaks are traditionally very difficult to detect and track down in any language. Java makes this even more so because of the non-deterministic nature of the Garbage Collector. In some cases, memory leaks only occur under particular circumstances that are difficult to reproduce in a test environment. Therefore, being able to track memory leaks in production can be vital to fixing the problem. With the proper tools, a memory leak can be quickly isolated and

the offending application can be managed while developers work on solving the problem.

4.2.5 Example 1: Identify Bad User

Periodically, the response time alert for the quick category of a URI's alert switches to a danger condition and then returns to the normal state. While random response time spikes can be expected in many production environments, examination of Figure 4-33 shows that the slow HTTP requests all originate from the same URI: /OrderStatus/OSController.

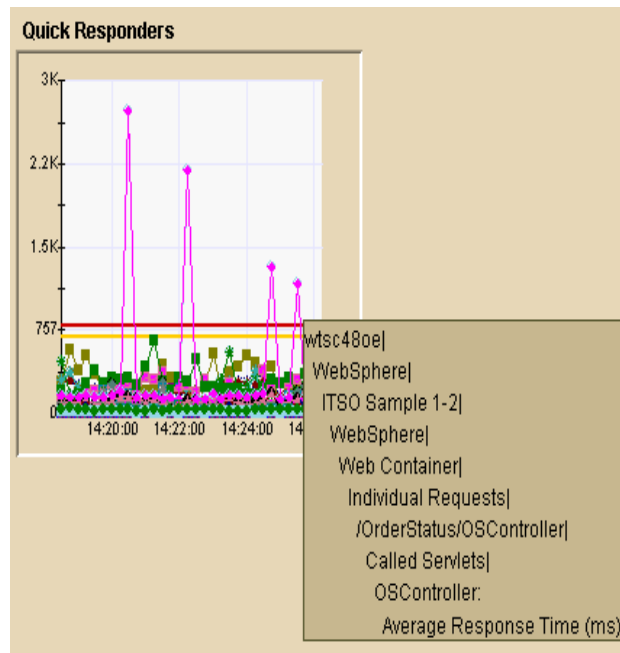


Figure 4-33 All slow response times are from the same URI

With no other information to go on and no strong intuition that could lead to a hypothesis, we guess that the database may be slow for a particular query. Over the same time period, Figure 4-34 on page 141 indicates that there is a query that is taking nearly twice as long as the next longest query. This figure reports SQL queries' average *round trip* time. Introscope defines query round trip time as the elapsed time required to execute the query and process the result set.

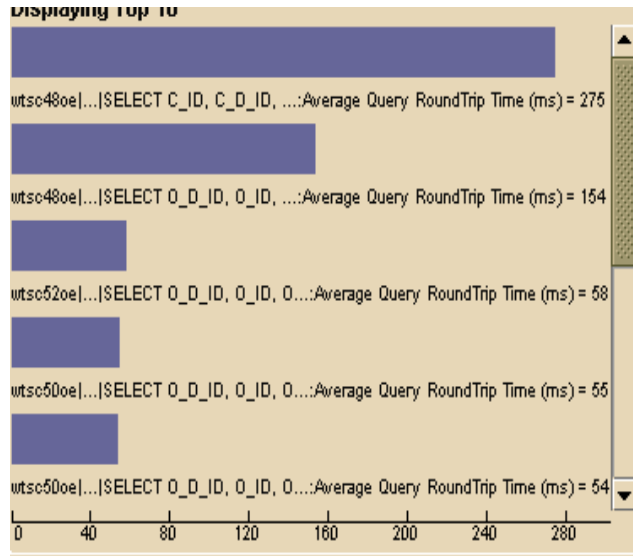


Figure 4-34 One query appears to be longer than the rest over a similar time frame

Now we have some suspicion that the problem is database-related, but we need to investigate further.

Figure 4-35 on page 142 illustrates using the Introscope Explorer to drill down and find what code is executing this query, in this case the `ejbFindCustomerByLastName` method of the `CustomerEntityBean`.

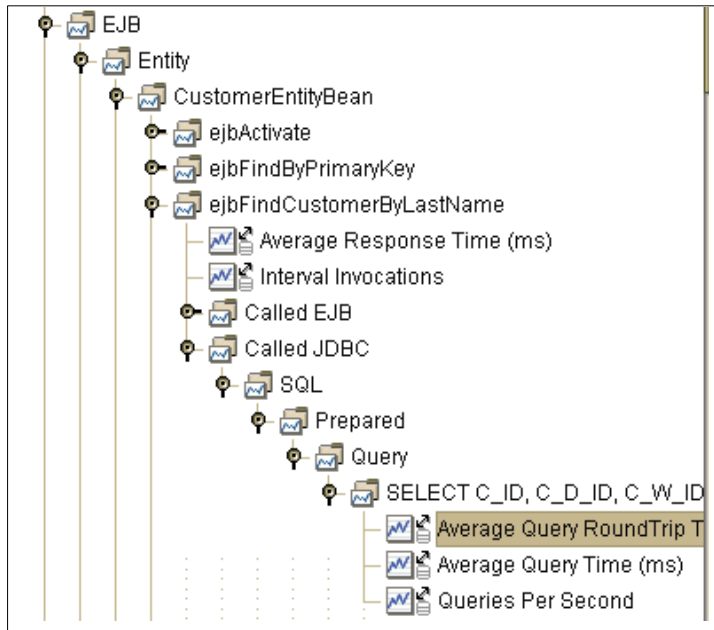


Figure 4-35 T Introscope Explorer shows that the `ejbFindCustomerByLastName` calls the badly performing query

Figure 4-36 shows the response time graph of this query's average round trip time over several successive 15-second intervals.



Figure 4-36 The average query round trip time for the SQL statement in question

Viewing the query's execute time in Figure 4-37 on page 143, we find that DB2 executes the query in less than one millisecond. Clearly the problem is not related to the DB2 query optimizer.

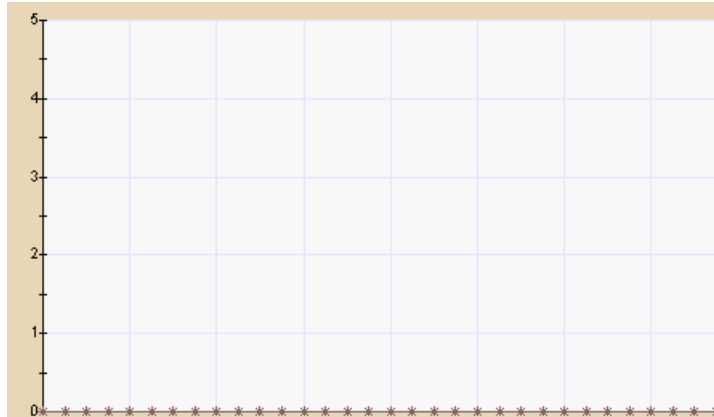


Figure 4-37 The same query's execute time is below 1 ms

Since this query's round trip time is so much longer than other queries (again, see Figure 4-34 on page 141), it is quite possible that the application code itself is to blame for the poor response.

In hopes of obtaining more information, we initiate a Transaction Tracer session to catch the individual WebSphere transactions performing poorly. Now we clearly see the source of the problem in Figure 4-38 on page 144. Examining each of the slow requests, we see that they have the same user ID, MIN. Going to the DBA with the query and the user ID, we discover that this particular user ID has several thousand rows in the table. Furthermore, the particular query that the application uses to retrieve the user information is poorly written, returning each of these rows when it should be specifying a more complete WHERE clause.

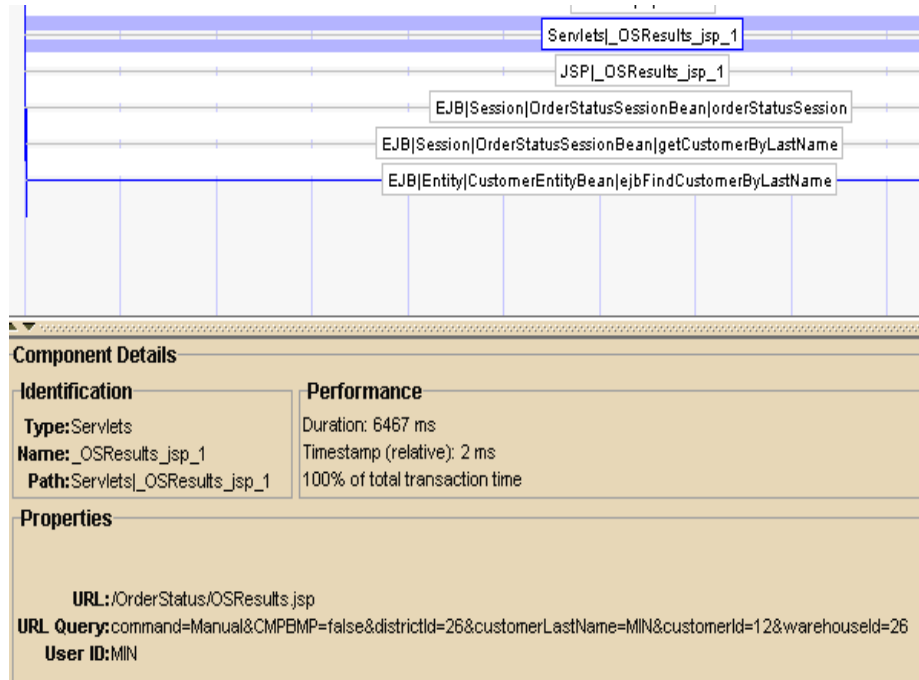


Figure 4-38 Selecting any servlet or JSP will show the user ID associated with the request

Conclusion

In some cases, the parameters of an individual transaction can cause poor response time. In this example, it was the user ID. The ability to trace an individual transaction with its parameters through the entire WebSphere server region and view the components contributing to its response time is critical for tracking down certain performance problems.

4.2.6 Example 7: Transaction Hang



Figure 4-39 Stalled Request Alert has turned red

In Figure 4-39 on page 144 we see that Introscope has alerted us that at least one in-flight WebSphere request has not responded within 30 seconds. Looking more closely at the stalled requests, Figure 4-40 shows that six requests within one server region are hanging on the MQSeries *queue get* operation. In fact, looking at other server regions, we see that most have one or two requests hung on this same operation.

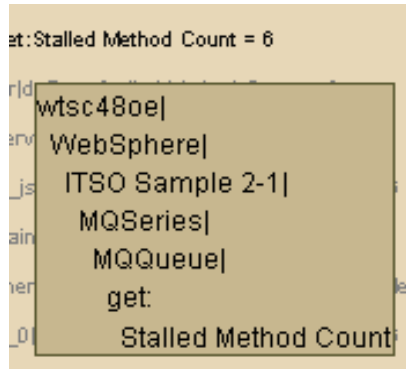


Figure 4-40 Six requests in this server region have hung on the MQSeries get operation

It turns out that the application relies on another application that, until now, was unknown to the production support team. This other application is supposed to place messages onto the MQ Queue for our WebSphere application to process. In this case, it stopped placing messages on the queue. Our application did not have a time-out facility, so it hangs indefinitely. In this configuration, each WebSphere server region only has six threads to handle incoming requests. Under these circumstances, one server region is entirely useless and must be restarted.

Conclusion

A broken MQ Series queue process is only one of several conditions that could cause a WebSphere transaction to hang. Poorly designed application e-mail components, external servers, and third-party systems are all potential sources of hung transactions. Making the problem worse, the J2EE specification can make it difficult for the application itself to provide appropriate facilities to time-out external requests. Therefore, it is critical that application monitoring include the ability to detect hung requests.

4.2.7 Example 8: Static Pages

We want to detect when WebSphere is serving static pages, so we configure Introscope to group all URLs dispatched by WebSphere that end with HTML, JPG, or GIF. This group, and any other arbitrary groups of performance metrics,

is configured interactively through the Introscope Workstation. The resulting group of metrics is shown in Figure 4-41.

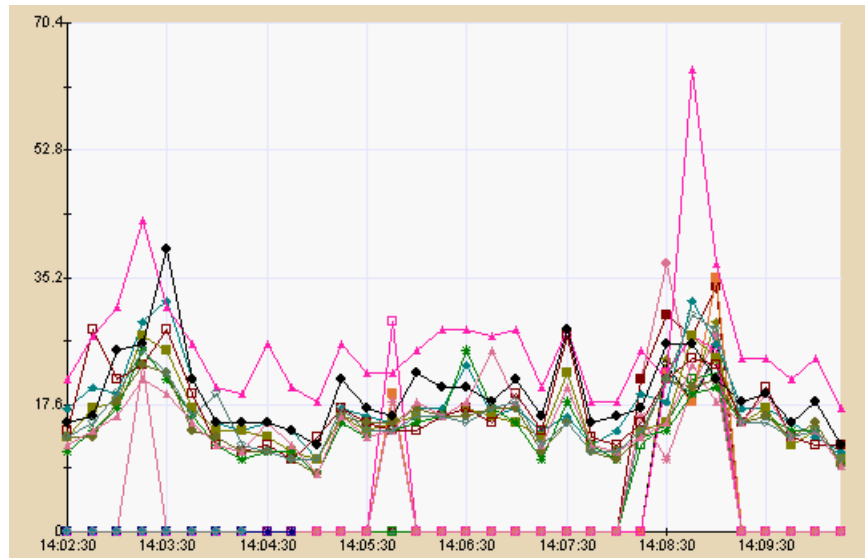


Figure 4-41 All dispatched URIs ending in HTML, JPG, or GIF

To find the component serving this static content, we switch to the Introscope Explorer and drill down on one of these dispatched URIs. Figure 4-42 on page 147 shows that the `/WebSphereSamples/TradeSample/TradeDocs/contentHome.html` URI is ultimately served by `SimpleFileServlet`. Exploring the other static content URIs yields the same result.

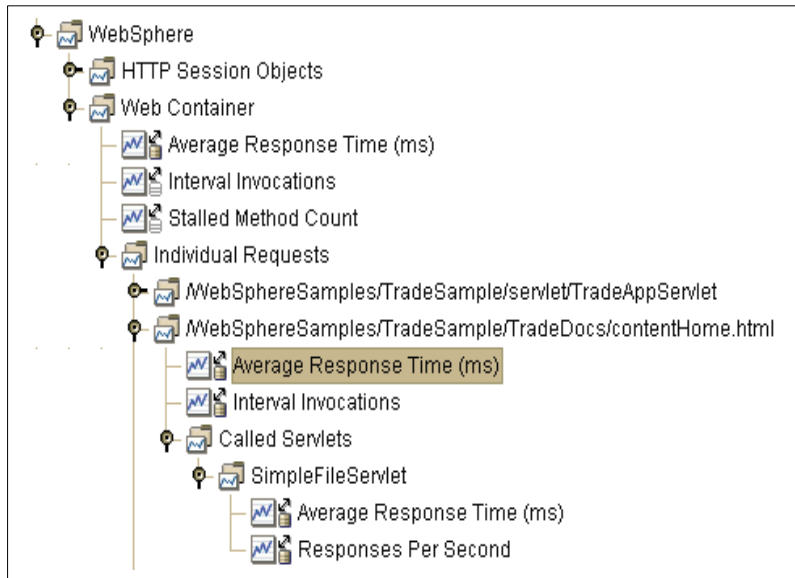


Figure 4-42 Using the Introscope Explorer's drill-down interface, we find that *SimpleFileServlet* is serving all static content

We want to know how much time WebSphere is spending serving this static content. We instruct Introscope to compute the average response time across all static content URIs. Then we instruct Introscope to compute the total number of static content URIs dispatched by WebSphere per second. The results of both of these computations are displayed in Figure 4-43 and Figure 4-44 on page 148, respectively.

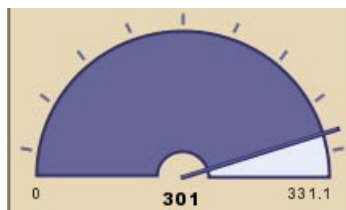


Figure 4-43 Total number of static content URIs dispatched by WebSphere per second

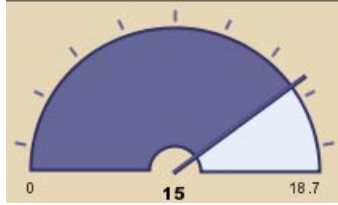


Figure 4-44 Average time WebSphere spends serving each static content URI

Multiplying the two yields the total number of milliseconds WebSphere spends serving static content during any particular time interval. In this case, approximately 4.5 seconds are used every 15 seconds to serve static content.

Conclusion

Many WebSphere installations choose to serve static content from a caching server such as IBM WebSphere Edge Server. Although this is not so much a throughput decision as it is an economic one, it is useful to be able to detect when WebSphere on z/OS is inappropriately serving static content because of a caching server failure or misconfiguration.

4.2.8 Example 10: Increased WebSphere Activity

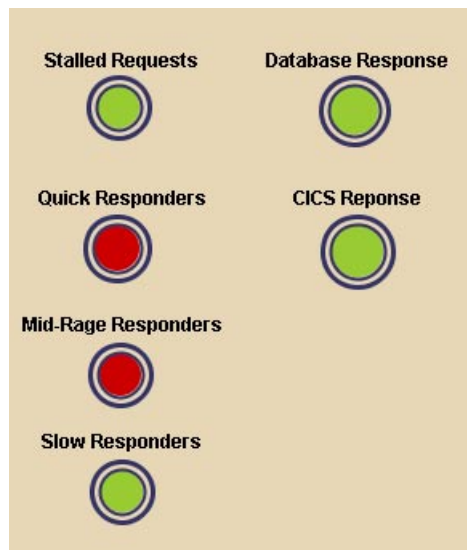


Figure 4-45 System is slowing down, but subsystem responses are good

In Figure 4-45 on page 148, we see that two of the three responder categories are responding poorly, but the corresponding subsystems are performing normally. Taking a closer look at the slow and mid-range responder categories in Figure 4-46, we see that many, if not all, of the HTTP requests are above their respective alerting thresholds. Figure 4-47 on page 150 explains why we see a green light in the slow responder alert: WebSphere hasn't received any requests in this category for a number of minutes.

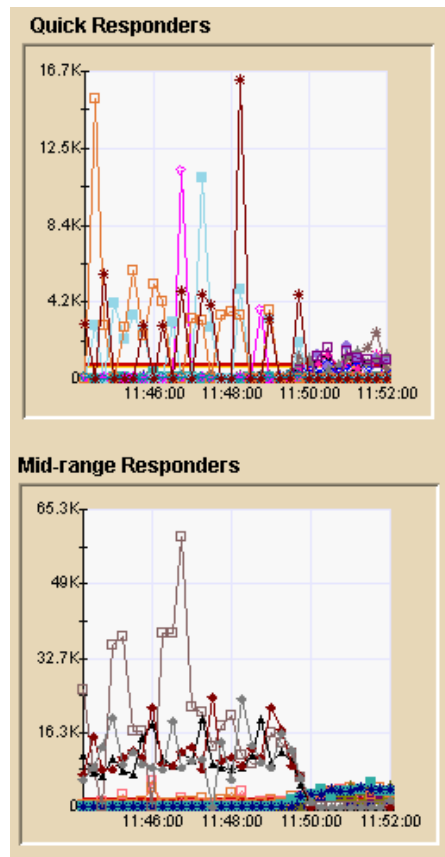


Figure 4-46 Many responses are well above alerting thresholds

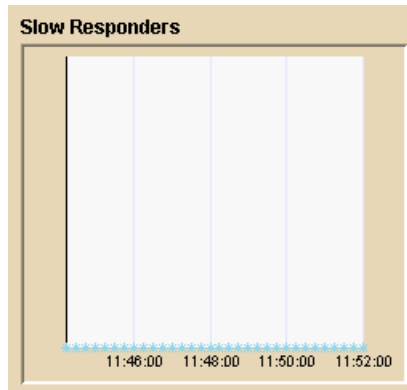


Figure 4-47 There are no requests in the slow responder category

Puzzled why response times would be slow, we take a closer look at the database in Figure 4-48 on page 151. We can see that, in a few cases, the database has responded between 1000 ms and 1400 ms, but this is rare. Most of its responses are well under the alerting thresholds. The other potential problem with database access is the time required to obtain a connection to the database. We can see that these response times are well under 100 ms.

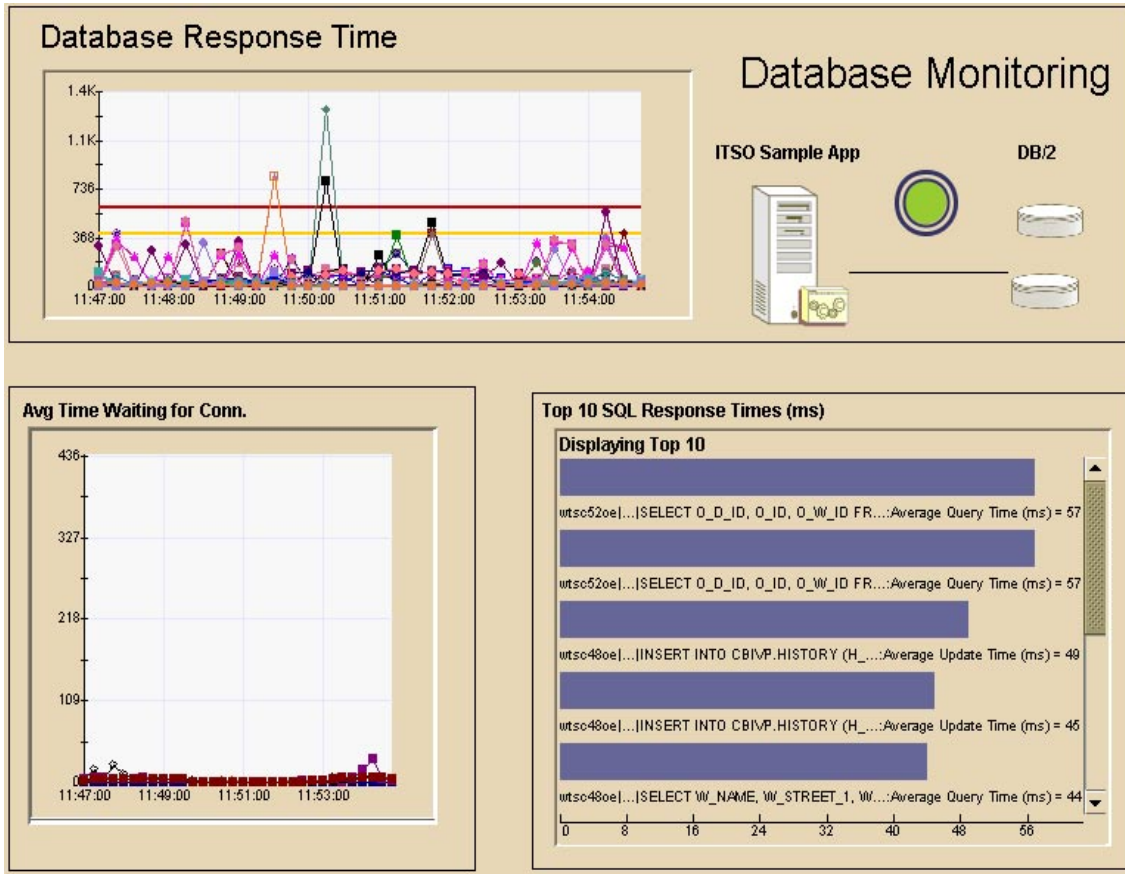


Figure 4-48 Most database-related accesses look good

Looking at the details does not shed any light on this problem. We decide to take a more global view of the entire sysplex. Figure 4-49 on page 152 illustrates the source of the problem: Aggregate load across all WebSphere server regions in the sysplex has been steadily increasing. The effect on response time is illustrated in Figure 4-49 on page 152. This graph represents the number of request services over all WebSphere server regions during a 15-second interval.

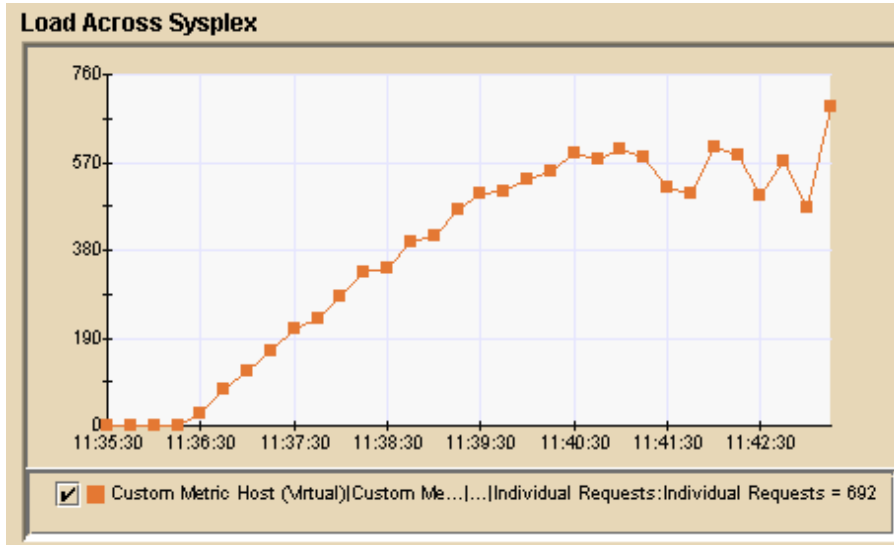


Figure 4-49 Load across the sysplex has been steadily increasing. The y-axis represents the number of individual requests WebSphere has received during a 15-second interval.

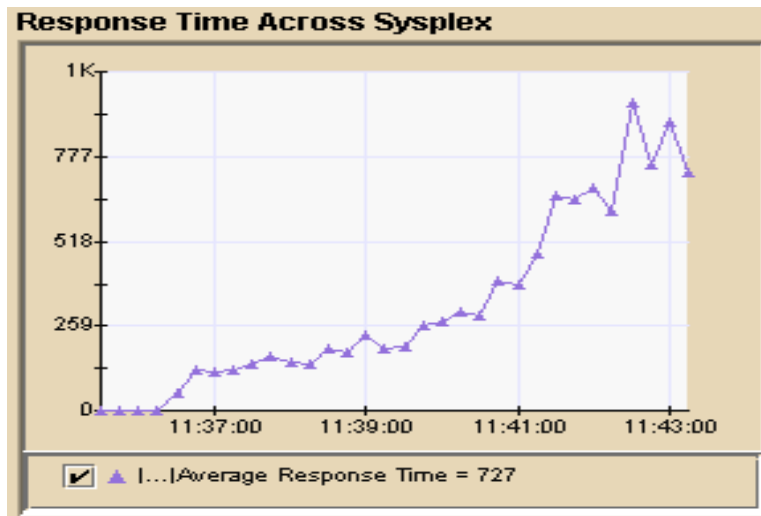


Figure 4-50 Average response times for all HTTP requests across the sysplex are increasing during the same time period as load has been increasing

We can see that average response time across the sysplex started to increase non-linearly at approximately 11:41. At the same time, we see from the graph of

sysplex load, the system handled 570 requests during a 15-second interval or 38 transactions per second. This is the so-called “knee of the curve,” the point at which the currently available system resources cannot bear more load without incurring substantially longer response times due to queuing.

Conclusion

Every system has a breaking point, and one way to find it is through experience. That experience can either come through testing the application or in production. Assuming we know the breaking point, appropriate throughput alert thresholds can be set to notify operations management that the application is beginning to become overloaded.

4.2.9 Example 11: Prioritizing Problems

In Figure 4-51, we see that four Introscope alerts are in the red (danger) state: CICS is responding slowly, DB2 is responding slowly, at least one request is stalled within a server region, and some of the requests in the mid-range category are responding somewhat more slowly than normal.

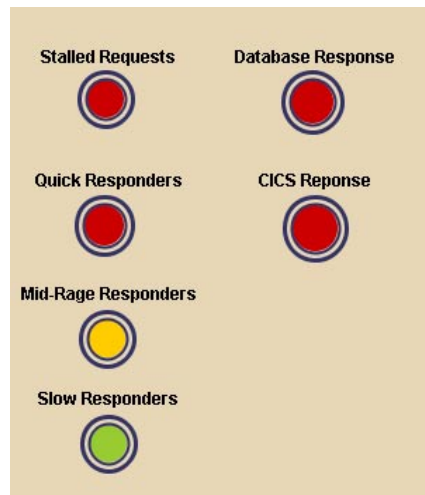


Figure 4-51 Many alerts are in the danger state

The immediate question is which alert to respond to first. In general, the answer depends on the business priority of the affected applications' functions and the degree to which the problem is perceived by end users. In this case, we know that stalled requests can eventually crash an entire server region, affecting all applications and all of the applications' users. Thus, the stalled request would seem to be the most important alert to attend to first. However, before beginning

that investigation, we take a quick look at the end-user response times to ensure that these problems aren't so bad as to warrant investigation first.

Figure 4-52 shows the response times across all categories. We see that most response times are either just above the danger threshold or below it. Clearly one of the requests in the “quick” responder category is well above its alerting threshold, with response times approaching 19 seconds.

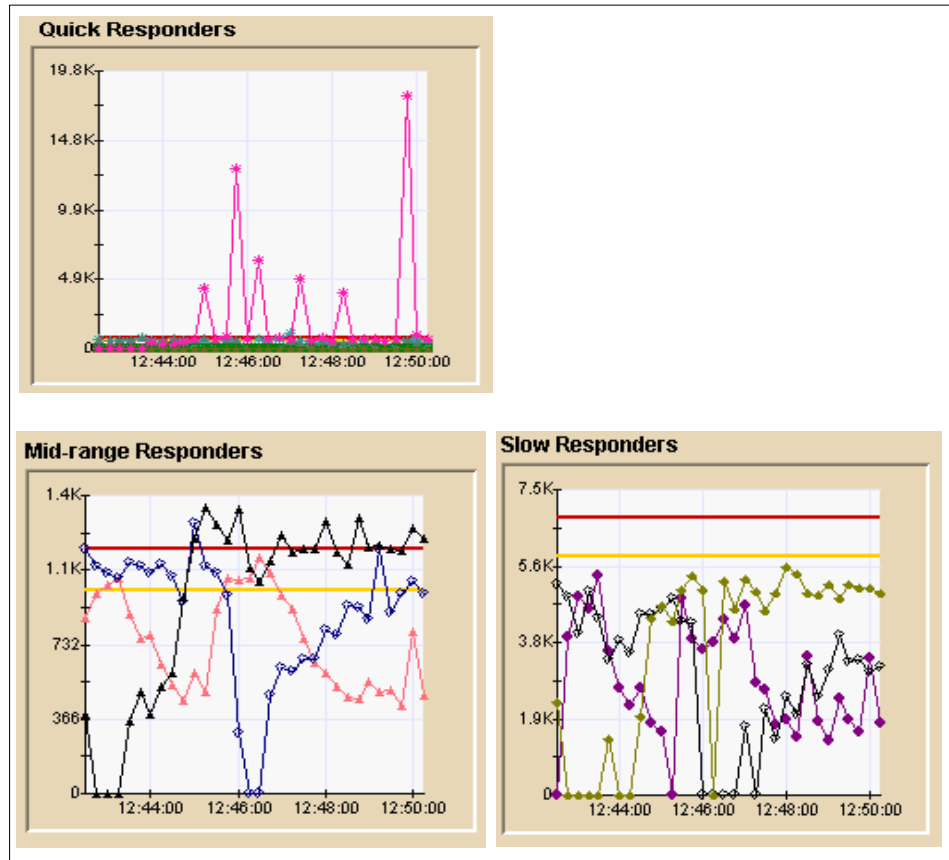


Figure 4-52 Response times for all response time categories. Most are ok, but one quick responder appears to be very bad.

Although response times this long are alarming, particularly when the normal response times for “quick” responders is below 800 ms, we simply note the problem for now and proceed to the problem that could crash the server: the stalled request alert.

An example of investigating a stalled request is described in 4.2.6, “Example 7: Transaction Hang” on page 144. In this case, we see that only one request is

stalled thus far. Because we have five more threads available and the number of stalled requests does not appear to be increasing, we decide to keep an eye on the stalled requests and proceed to investigate the other alerts.

Again, the choice as to which problem to track down first ideally involves the relative business value of the requests affected. Although we know from Figure 4-52 on page 154 which HTTP requests are slower than normal, we have no information to weigh their relative business values. Therefore, we must choose using a different criterion.

In this case, we know that the database is involved in many more requests than CICS so we decide to investigate it first. Examples of investigating database problems can be found in sections 4.2.2, “Example 6: No DB2 Index” on page 129 and 4.2.5, “Example 1: Identify Bad User” on page 140. After we track down that problem, we can investigate CICS (an example of which can be found in 4.2.1, “Example 4: CICS” on page 126).

In real life, multiple problems can occur simultaneously, and we have no way of knowing whether the problems are related to one another. So, we must have some order for investigating the problems and that generally implies prioritizing the worst problems first. Ideally, the worst problems would be those that most greatly affect the users of the application. Unfortunately, that requires assigning a business value to the requests entering WebSphere and that information is rarely communicated to operations personnel. Therefore, we are left with our experience and general knowledge of the application to set the priorities.



PathWAI solutions for WebSphere

Candle's PathWAI™ solutions are built on more than 26 years of experience managing mission-critical applications with the OMEGAMON® performance monitor. PathWAI solutions consist of packages of software, services, and training. These packages are tailored to suit each phase of the “build, deploy, manage” life-cycle of WebSphere e-business initiatives.

5.1 PathWAI solutions

PathWAI packages offered for WebSphere Application Server include:

- ▶ PathWAI Architecture for WebSphere provides architecture assessment for the design phase.
- ▶ PathWAI Deployment for WebSphere provides performance tuning to ensure scalability for the deployment phase.
- ▶ PathWAI Monitor for WebSphere Application Server provides performance monitoring tools and customized services to manage development and test environments.
- ▶ PathWAI Dashboard for WebSphere Infrastructure provides an integrated performance monitoring solution to manage production environments.

5.2 OMEGAMON XE performance monitors

PathWAI Monitor and Dashboard core packages include the OMEGAMON XE for the WebSphere Application Server performance monitor. Additional performance monitors for connected applications and platforms can be added to the PathWAI packages.

The PathWAI Dashboard solution also includes OMEGAMON DE technology, which enables you to monitor all the components of your WebSphere infrastructure from a single “dashboard view” of the enterprise. In this dashboard view you can combine performance metrics from a variety of applications on disparate platforms.

For example, you can see performance metrics from WebSphere Application Server, WebSphere MQ, DB2 and the underlying operating system in the Dashboard view. These reports are browser-accessible and can be personalized to monitor the metrics that are most critical to your enterprise. For example, you could display JVM memory usage and response times for your WebSphere Application Server JSPs and servlets, and monitor the “dead letter queue” for WebSphere MQ from a single “pane of glass”. OMEGAMON's event-based monitoring and historical trend analysis can help you address performance problems before they cause slowdowns or downtime.

5.2.1 OMEGAMON XE architecture

OMEGAMON XE provides a multi-tiered, multi-platform, client-server architecture for maximum flexibility and scalability.

Figure 5-1 depicts a high-level view of the architecture showing the OMEGAMON XE clients, servers and agents.

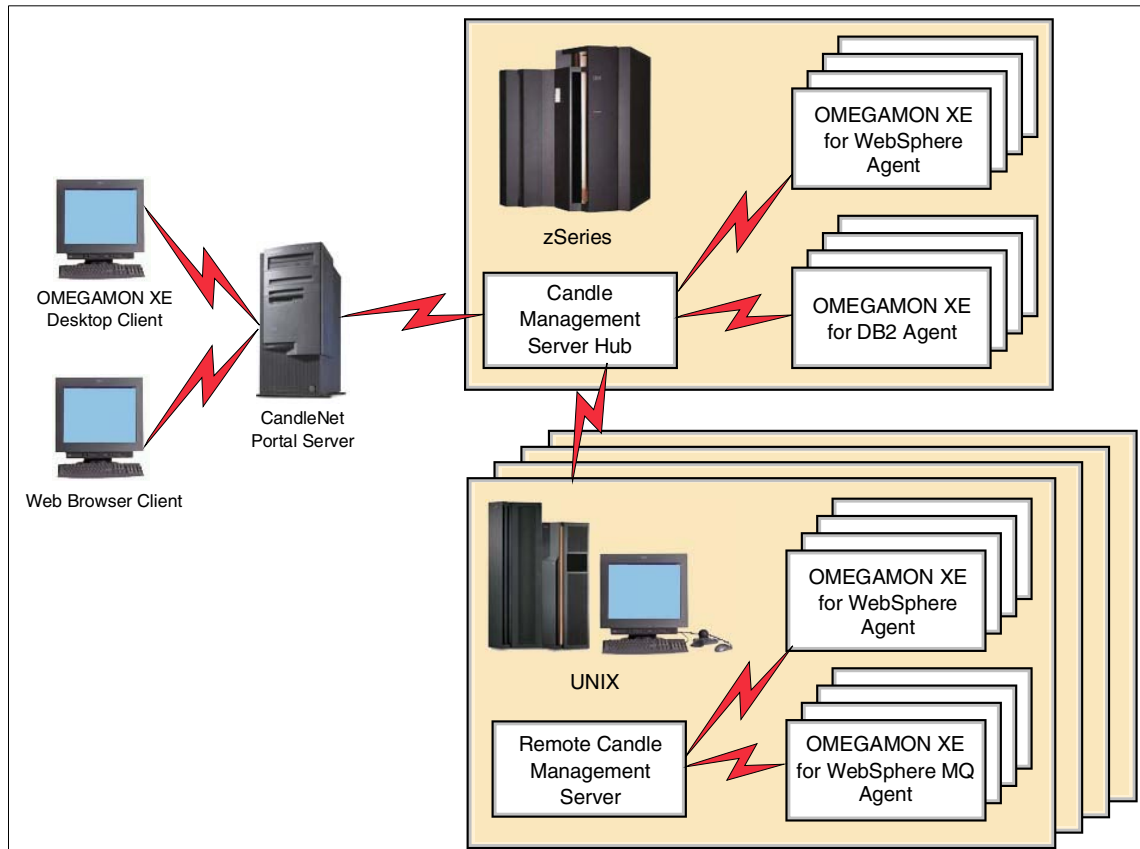


Figure 5-1 OMEGAMON XE architecture

Clients

OMEGAMON XE provides an easy-to-use Java-based client that runs on a workstation as a desktop application or accessed through a Web browser.

The OMEGAMON XE client displays status information for monitored components using red, yellow, or green indicators, denoting critical, warning, or normal status, respectively. The client also provides access to real-time and historical data for problem analysis, and facilitates user administration, event definition, and automation.

The OMEGAMON XE screens, known as *workspaces*, are comprised of multiple views. By default, each workspace includes a navigator view that shows status from the Candle monitoring agents organized by the operating system platform in a tree-like hierarchy. Figure 5-2 shows a sample OMEGAMON XE workspace.

The navigator tree view includes agents on multiple UNIX and Windows servers organized by platform.

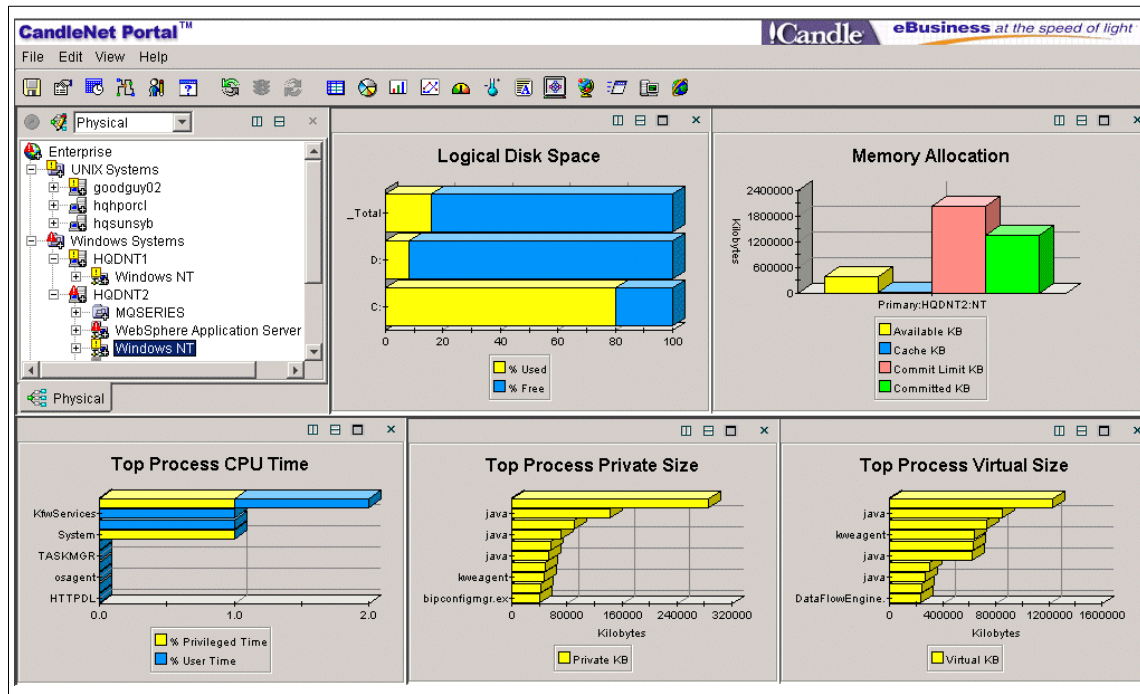


Figure 5-2 Sample OMEGAMON XE workspace

Feature highlights of the OMEGAMON XE interface are:

- ▶ Easily customizable to present data in a variety of formats, including bar charts, graphs, tables and pie charts
- ▶ Easily switch from real-time data to historical views
- ▶ Access Web applications within the OMEGAMON XE interface
- ▶ Access mainframe applications using the 3270 and 5250 terminal emulator views
- ▶ Display the status of your enterprise by physical system, by geographic location on a map, or create a business view to represent application components in a logical view
- ▶ Issue console commands and invoke product-provided actions to dynamically tune monitored components

In addition, with OMEGAMON DE, each view can contain performance and availability data from a different OMEGAMON XE monitoring agent for an integrated overview of your WebSphere enterprise. OMEGAMON XE clients run on Windows NT, Windows 98, Windows 2000 and Windows XP Professional.

The OMEGAMON XE clients are connected to the CandleNet Portal Server.

CandleNet Portal Server

The CandleNet Portal Server (CNPS) services requests from the OMEGAMON XE clients. Its function is to collect, analyze and format the data for presentation by the OMEGAMON XE client. The CNPS runs on Windows NT, Windows 2000, or Windows XP Professional.

The CNPS is connected to the Candle Management Server hub.

Candle Management Server

The Candle Management Server® (CMS™) acts as the hub for the monitoring agents. It collects performance and availability data from the agents and passes it to the CandleNet Portal® Server. The CMS also evaluates Candle-provided and user-defined situations, to determine whether a threshold has been exceeded or a condition has been met. When a situation is true, an alert is displayed on the OMEGAMON XE navigator tree view. Situations can also be automated to take corrective actions or provide problem notification by pager, e-mail, or voice application.

Secondary Candle Management Servers (remote CMS servers) may be added to provide scalability and load balancing. You can also configure a hot-standby CMS that acts as a backup in a fault-tolerant environment. CMS runs on z/OS, OS/390, UNIX, Windows NT, or Windows 2000.

CMS is connected to one or more *agents*.

Agents

OMEGAMON XE agents monitor applications, databases, subsystems, operating systems, etc. They must be installed on the same node or LPAR that hosts the resources to be monitored.

Agents collect data upon request from the hub or remote CMS to which they are connected.

Agents support requests for data as follows:

- ▶ Satisfy real-time requests to display data in OMEGAMON XE workspaces.
- ▶ Provide data for situation evaluation on a user-defined interval.
- ▶ Collect historical data based on a user-defined interval.

Alert managers are specialized agents that monitor alerts from a third-party product or send data about events monitored by Candle products to a third-party

management application, such as Tivoli Enterprise™ Console or CA Unicenter TNG.

OMEGAMON XE monitoring agents are available for many hardware platforms, operating systems and applications, as shown in Table 5-1.

Table 5-1 *Candle agents*

Platform	Monitored applications
zSeries	CICS, Crypto, DB2, IMS, OS/390 UNIX System Services, Mainframe Networks, OS/390, storage, sysplex
Middleware	WebSphere MQ, WebSphere MQ Integrator
Distributed platforms	Linux, OS/400®, UNIX, Windows NT
Distributed applications	DB2 UDB, MS SQL Server, Netware, Oracle, R/3, Sybase, Tuxedo
Application end-user response	SAP, Citrix MetaFrame Server

Additionally, the Candle Universal Agent is a generic agent that allows you to integrate and monitor any type of data that is collected at your site into the PathWAI solution.

5.2.2 Monitoring WebSphere Application Server

OMEGAMON XE for WebSphere Application Server provides the following performance monitoring capabilities for the entire WebSphere application life cycle:

- ▶ Workload analysis
- ▶ Application trace
- ▶ SMF information
- ▶ JVM profiler data
- ▶ Configuration and environmental data

Workload analysis

Workload analysis is a powerful feature that quickly identifies resource bottlenecks affecting your WebSphere applications. This feature measures the response time of servlets, JSPs, and EJB methods, and identifies where these workloads are spending their time.

All Workloads is the main workspace provided by workload analysis. For each workload you can see the average response time, CPU time, number of invocations, and a breakdown of the response time by component. For example, the amount of time spent performing JMS requests, JNDI lookups, or SQL updates. The ten workloads with the worst average response times are represented in a bar chart, with the response time for each component displayed in a different color.

Figure 5-3 is an example of the All Workloads workspace for the Trade2 sample application.

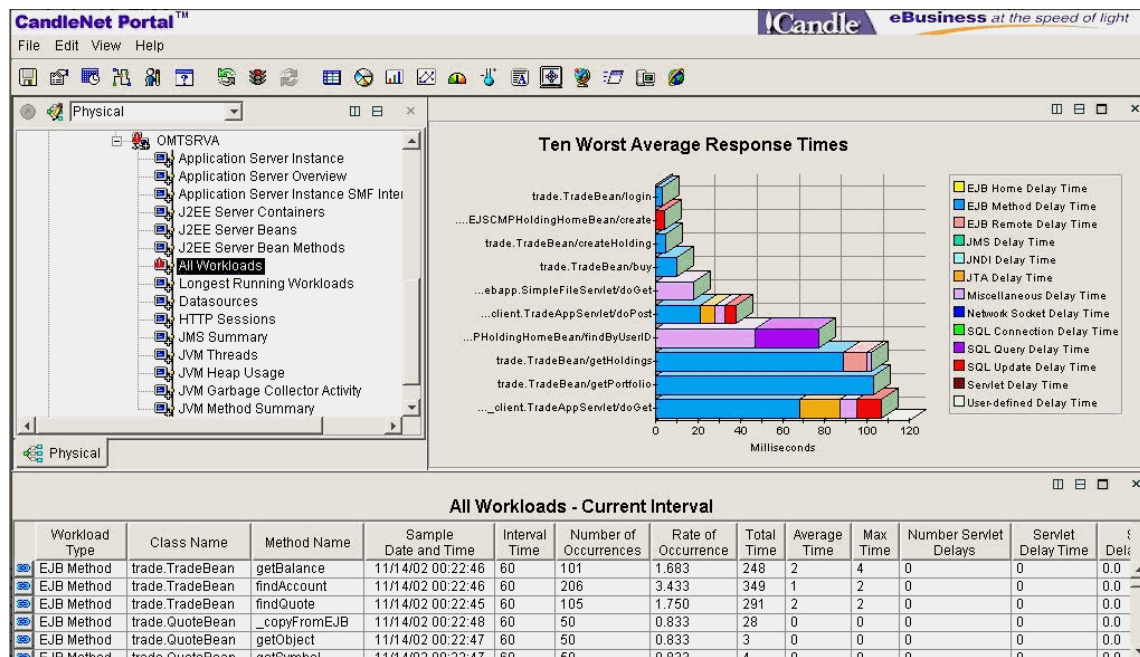


Figure 5-3 All Workloads workspace

Workload analysis is suited for all stages of application development, deployment, integration testing, and production monitoring. You control which servlets, JSPs and EJB methods are instrumented at start-up, then dynamically control the collection of workload analysis data at runtime. Workload analysis data can be displayed by workload or by resource.

Table 5-2 on page 164 provides a high-level summary of the major workload analysis workspaces.

Table 5-2 Workload analysis workspaces

Workspace	Function
All Workloads	Displays average response times by workload. Graphically breaks down the response times into delay components. Use this workspace to detect poor response times and bottlenecks for business applications.
Selected Workload Delays	Displays detailed information for a delay component (JMS, JNDI, SQL, etc.) related to an individual workload. For example, displays average response times for each SQL update request made by a workload.
Longest Running Workloads	Displays response times for individual invocations of each servlet, EJB method, and JSP that exceeds a response time threshold. Graphically breaks down the individual response times into delay components. Use this workspace to identify users receiving the worst response times, when the average response is good.
Datasources	Displays details about each J2EE datasource accessed by all workloads. Graphs average connection wait time, and processing times for SQL queries and SQL updates for each J2EE datasource. Use this workspace to determine whether you have a bottleneck in your DB2 regions.
HTTP Sessions	Displays information on the current active HTTP sessions including the Web application name, IP address, and userid. Graphically displays the number of HTTP sessions during the last four hours. Use this workspace to monitor throughput.
JMS Summary	Displays details about each WebSphere MQ queue accessed by all workloads. Graphs the average response times for browse, put, and get requests for each queue. Use this workspace to determine whether you have a bottleneck in WebSphere MQ.

Additional workspaces provide alternative formatting for workload analysis data for the current interval or across historical collection intervals.

Application trace

The application trace feature provides the ability to trace the inter-method flow within an application.

Ideally suited for application development and testing, this feature utilizes the instrumentation that is installed at start-up for workload analysis. Traces are dynamically started at runtime for specific workloads. The trace file contains an

entry for each method call, method return, workload start, workload end, and thrown exception in the call flow for the workload.

Refer to Figure 5-19 on page 181 for an example of the Application Trace workspace.

SMF Information

OMEGAMON XE for WebSphere Application Server provides a convenient way to format SMF 120 interval records for J2EE and managed-object framework (MOFW) servers.

This feature can be used in any environment where SMF 120 interval records have been enabled. OMEGAMON XE intercepts the SMF records as they are being written and extracts data for the classes, beans, and methods you select.

Using the formatting features of OMEGAMON XE, you can easily generate customized views using filtering and sorting. You can also plot SMF data across historical intervals.

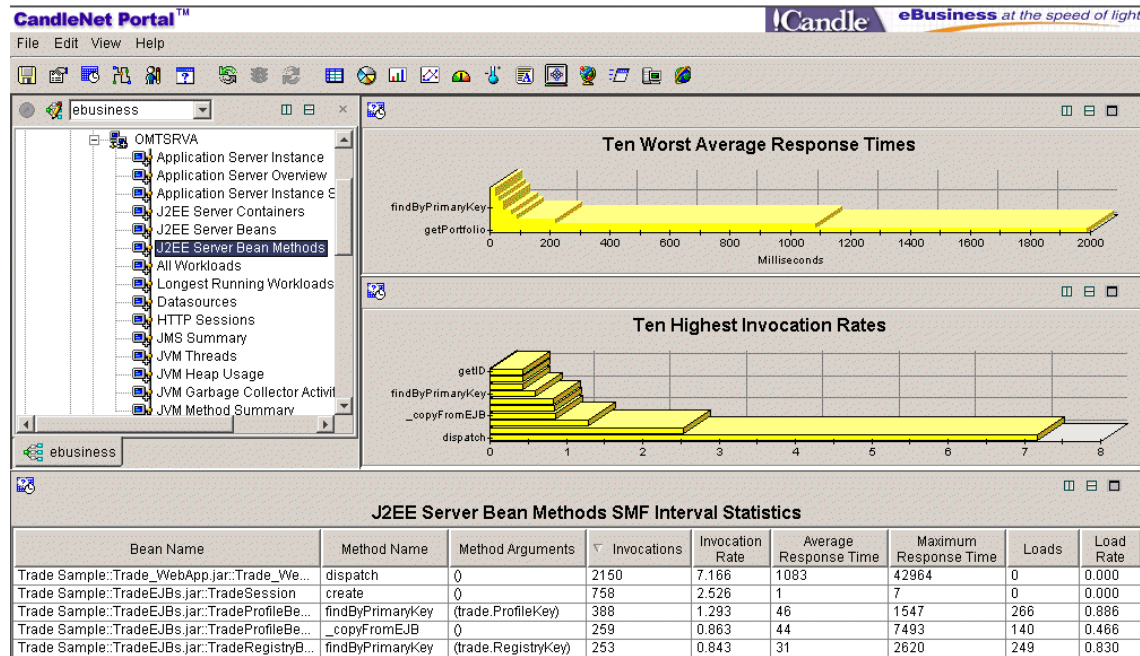


Figure 5-4 SMF Interval statistics for methods

For example, Figure 5-4 shows the J2EE Server Bean Methods SMF Interval Statistics workspace. This workspace graphs the ten methods with the worst

average response time, and the ten methods with the highest invocation rate during the SMF interval. This enables you to see at a glance whether your most frequently requested methods are responding poorly.

JVM profiler information

OMEGAMON XE for WebSphere Application Server provides detailed performance information from the JVM profiler interface (JVMPPI).

This feature is best suited for development and test environments. The JVMPPI must be loaded at WebSphere start-up, then you control collection dynamically at runtime. This feature does not require instrumentation of your applications.

JVMPPI provides the following workspaces:

- ▶ JVM Garbage Collector Activity
- ▶ JVM Heap Usage
- ▶ JVM Threads
- ▶ JVM Methods
- ▶ JVM Monitor Contention
- ▶ JVM Method Summary

For examples of the JVM Garbage Collector Activity and the JVM Heap Usage workspaces, refer to Figure 5-26 on page 187 and Figure 5-27 on page 188.

Configuration and environmental data

OMEGAMON XE for WebSphere Application Server provides workspaces that display configuration data including environment variables and JVM properties. You can use these workspaces to determine whether your server instances are configured correctly:

The Application Server Error Logstream workspace displays error messages from the WebSphere logstreams. You can add situations to detect specific error message IDs and add automation to take corrective action. Using Candle's Alert Adapter™ for AF/REMOTE® you can set up intelligent scripts that can escalate problem conditions via audible alarms, pagers, third-party software, etc.

5.2.3 Monitoring the WebSphere environment

In 1.6, “Performance components” on page 18 we emphasize that this is a complex performance environment with many critical components.

The performance of your WebSphere-based applications is dependant on many components outside of the WebSphere Application Server. Candle offers OMEGAMON XE agents for all these critical components, which can be integrated using OMEGAMON DE to monitor your entire environment.

Monitoring the TCP/IP network

OMEGAMON XE for Mainframe Networks can monitor any application that has a TCP/IP connection open on z/OS. This information can be used to determine the volume of data being requested from WebSphere Application Server.

In 5.4.6, “Example 8 - Static pages serving” on page 207 we use OMEGAMON XE for Mainframe Networks to determine the impact on TCP/IP when the WebSphere Edge Server servers are not caching static files. Refer to Figure 5-63 on page 211 for an example of the Network Applications workspace monitoring the byte rate for WebSphere Application Server.

Monitoring zSeries hardware and z/OS

OMEGAMON XE for OS/390 provides features to monitor the health of z/OS hardware and software resources.

For example, OMEGAMON XE for OS/390 can be used to monitor the effective weighting of your LPARs.

Looking at the CPC LPAR Status view in Figure 5-5 on page 168 we see that production sysplex WTSCPLX1 is highlighted because it has an effective weight index of 0.8, compared to an index of 10.6 for SANDBOX. In this case, this is not a problem as the “logical weights” (definition) shows WTSCPLX1 as much higher; it just doesn't have any workload at the moment.

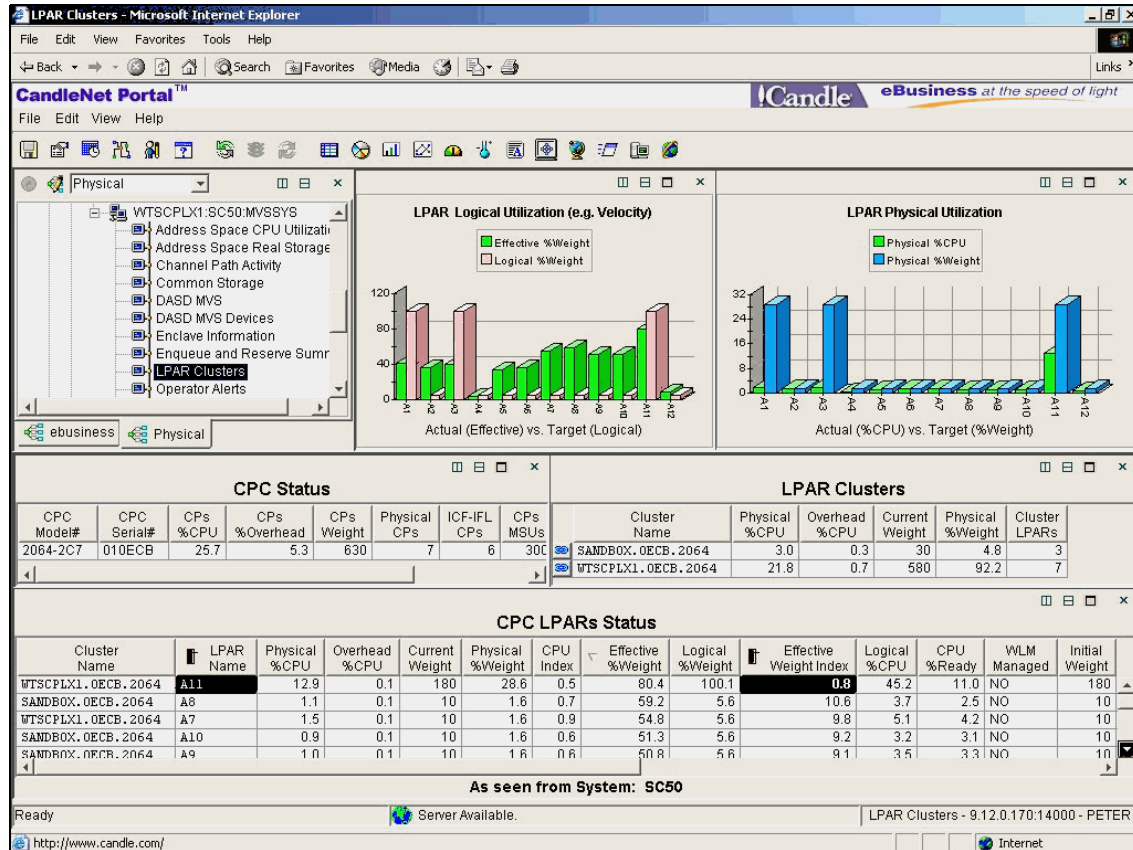


Figure 5-5 LPAR weights

Monitoring UNIX System Services

OMEGAMON XE for OS/390 UNIX System Services can monitor USS processes for availability, looping, or scaling.

For example, WLM can dynamically create WebSphere server regions to handle increased arrival rates. You can use OMEGAMON DE to monitor the number of WebSphere server regions from the USS agent relative to the transaction throughput from the WebSphere agent.

OMEGAMON XE for OS/390 UNIX System Services can also be used to monitor HFS I/O rates. We use this agent in 5.4.6, "Example 8 - Static pages serving" on page 207 to determine whether there was an increase in HFS I/O as a result of WebSphere Application Server handling static pages. Refer to Figure 5-62 on page 210 for an example of the OS/390 USS Mounted File Systems workspace.

Monitoring WorkLoad Manager

Ensuring that there are enough server instances and that each instance gets enough CPU and memory is the job of the Workload Manager (WLM). OMEGAMON XE for OS/390 allows you to determine whether your workload is allocated sufficient resources. OMEGAMON XE for Sysplex provides a sysplex-wide view of a service class so you can detect workload imbalances between z/OS images.

Monitoring RRS and Coupling Facility

WebSphere Application Server and DB2 use RRS, which uses Coupling Facility structures. In the ITSO configuration there are three DB2 subsystems in a DB2 data sharing group. DB2 data sharing has several critical structures in the Coupling Facility (global buffer pools, locks, and SCA). The latter two are critical to performance as they perform synchronous I/O, which causes the requesting CP to wait until the request is satisfied.

OMEGAMON XE for Sysplex enables you to verify that all members of the data sharing group are connected. You can also use this agent to alert you before the Coupling Facility structures fill up.

Monitoring security

OMEGAMON XE for Crypto monitors cryptographic coprocessors on z/900s. Configuration errors can result in co-processors not being used. Instead, the SSL is encrypted and decrypted via software, which uses CPU cycles and may result in workload slowdowns. None of the ITSO test examples have HTTPS or any SSL, so this product was not installed.

WebSphere uses UNIX Systems Services and forks lots of threads, each of which loads modules and potentially accesses the HFS. Access to these files requires SAF calls to RACF and has the potential to impact performance.

You can use OMEGAMON XE for OS/390 to monitor the RACF address space for excessive I/O and CPU.

Monitoring JVM

OMEGAMON XE for WebSphere Application Server provides detailed information on the JVM, including garbage collection cycles and heap usage. Refer to “JVM profiler information” on page 166 for more details.

Monitoring WebSphere

Refer to 5.2.2, “Monitoring WebSphere Application Server” on page 162 for information on the features provided by OMEGAMON XE for WebSphere Application Server to monitor WebSphere resources.

Monitoring connectors and subsystems

WebSphere Application Server can use connectors to obtain data from subsystems such as DB2, IMS, and CICS. It can also be connected to back-end systems via WebSphere MQ. OMEGAMON XE for WebSphere Application Server monitors JMS, JDBC and JCA requests, which enables you to monitor the use of connectors and MQ from a WebSphere perspective.

The following products enable you to monitor the use of connectors and MQ from the subsystem perspective, as well as providing detailed information on subsystem resources:

- ▶ OMEGAMON XE for CICSplex and OMEGAMON XE for CICS
- ▶ OMEGAMON XE for DB2plex and OMEGAMON XE for DB2
- ▶ OMEGAMON XE for IMSplex and OMEGAMON XE for IMS
- ▶ OMEGAMON XE for WebSphere MQ

Monitoring applications

Due to the complexity of the environment, there are many opportunities for design errors to impact performance. The root cause could be in the application or the system setup. We need to look at all the components, their connectivity and load balancing to determine the source of the problem. OMEGAMON DE enables you to integrate information from all the OMEGAMON XE products to provide a single point of control for determining the source of a problem.

5.3 PathWAI configuration at ITSO

PathWAI Dashboard for WebSphere Infrastructure was installed in the ITSO environment. This package is comprised of the following monitoring products:

- OMEGAMON XE for WebSphere Application Server
- OMEGAMON XE for OS/390 UNIX System Services
- OMEGAMON DE

The following monitoring products were also installed in the ITSO environment, based on the requirements of the test applications and environment:

- OMEGAMON XE for CICSplex
- OMEGAMON XE for DB2
- OMEGAMON XE for Mainframe Networks
- OMEGAMON XE for WebSphere MQ
- OMEGAMON XE for OS/390
- OMEGAMON XE for Sysplex

The agents were installed on all three LPARs (SC48, SC50 and SC52). The agents were connected to a hub CMS on LPAR SC48.

Customized navigator view

By default, OMEGAMON XE displays data from the agents in a navigator tree view organized by physical nodes. OMEGAMON DE provides the capability to create customized navigator views to integrate performance metrics from disparate platforms and components and map them to business applications.

We leveraged this capability to define an ebusiness navigator view that represents the three applications in the ITSO environment:

- ▶ Trade2, for the Trade 2 application on the OMTSRVx application server instances, as well as performance data for DB2, TCP/IP, USS, WLM, WebSphere MQ across the three LPARs
- ▶ Inventory Control, for the eITSO application on the OMESRVx application server instances, plus DB2, USS, and WebSphere MQ
- ▶ EIS, for the PRR application on the OMTSRVx application server instances, plus CICS, USS and WebSphere MQ

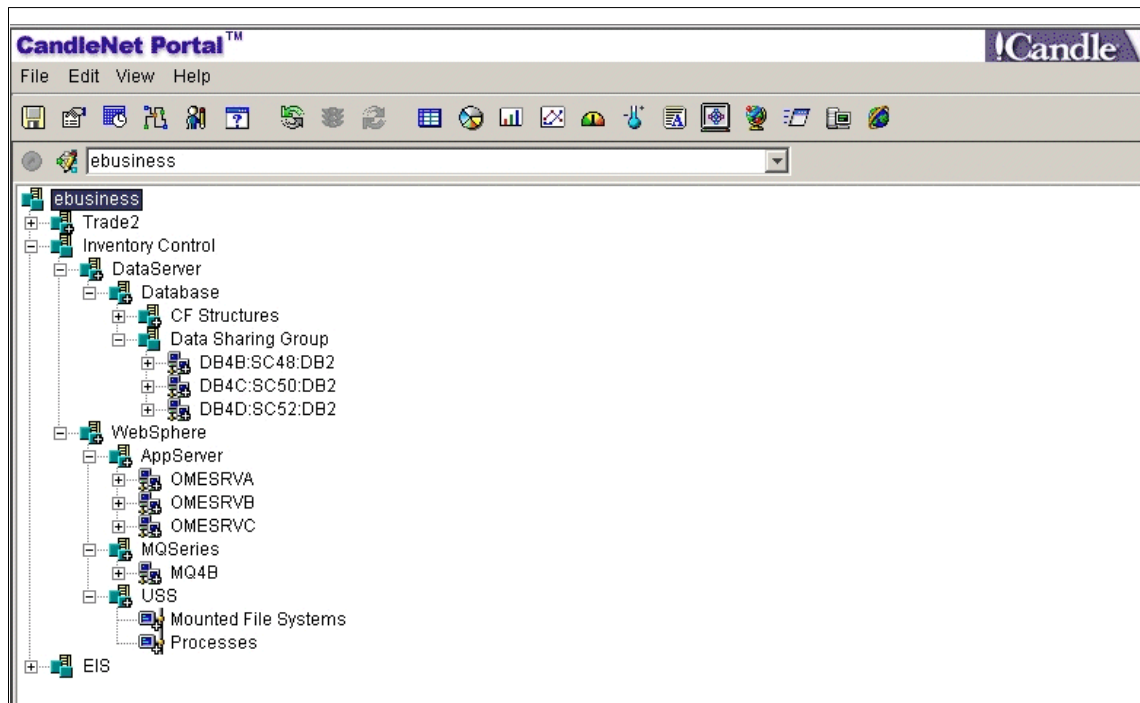


Figure 5-6 ebusiness navigator view

Figure 5-6 shows the eBusiness navigator view for the three applications, expanded to show details for Inventory Control. The navigator tree for Inventory Control includes the DB2 data sharing group, MQ queue manager, three WebSphere application server instances, and USS mounted file systems and processes across the three LPARs.

Customized business view

OMEGAMON DE provides the capability to display data from multiple agents running on multiple LPARs into a single integrated workspace. With an understanding of the PRR application, we created a customized workspace that allows us to proactively monitor and display key metrics from WebSphere Application Server, WebSphere MQ, CICS, and UNIX System Services (USS); see Figure 5-7 for details.

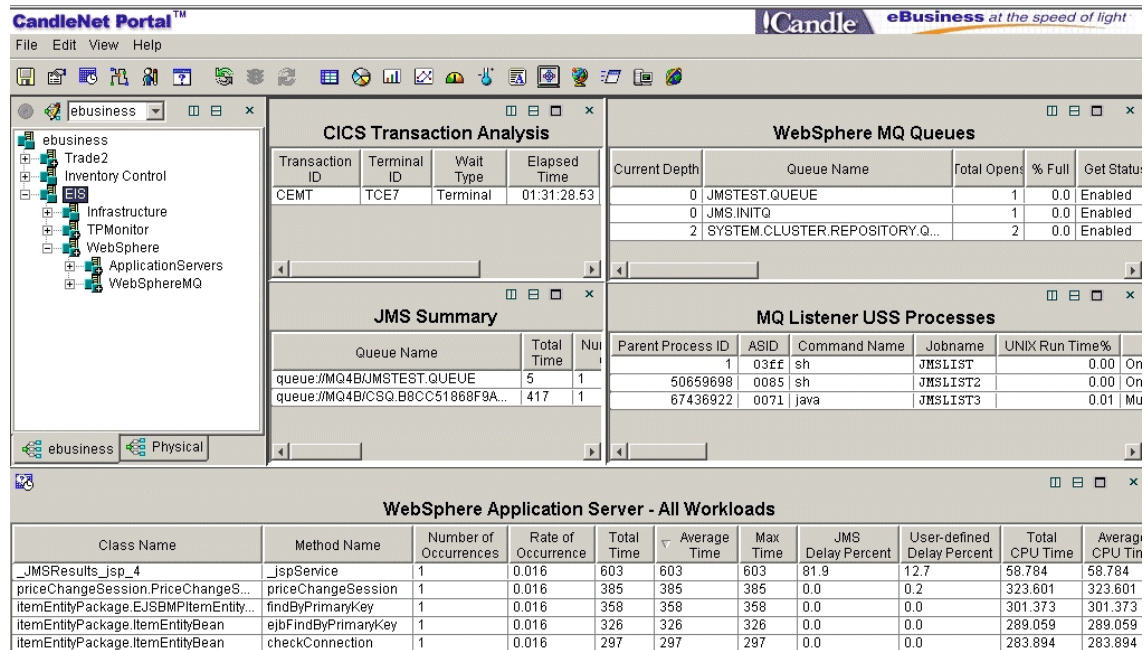


Figure 5-7 EIS business view

The CICS Transaction Analysis view displays task information for CICS region SCSCERW1. This region provides the CICS CTG support for application PRR. This information is obtained from OMEGAMON XE for CICSplex.

The WebSphere MQ Queues view displays information on queue JMSTEST.QUEUE from queue manager MQ4B used by the PRR application. This information is obtained from OMEGAMON XE for WebSphere MQ.

The *JMS Summary* view displays response time information for JMS requests for server instance OMTSRVA. This information is obtained from OMEGAMON XE for WebSphere Application Server.

The *MQ Listener USS Processes* view displays information on the USS processes that get the messages from the MQSeries trigger queue and return data on the reply-to queues for the PRR application. This information is obtained from OMEGAMON XE for OS/390 UNIX System Services.

The *WebSphere Application Server - All Workloads* view displays average response times for workloads on server instance OMTSRVA. This information is obtained from OMEGAMON XE for WebSphere Application Server.

Refer to 5.4.3, “Example 4 - Identify a CICS TS response time problem” on page 188 and 5.4.5, “Example 7 - Transaction hang or time-out” on page 200 on how this custom business view can be used to quickly diagnose problems across components.

Customized alerts

OMEGAMON XE provides the capability to set a warning and critical threshold for any performance metric. These situations generate an alert on the navigator tree view when a threshold is exceeded.

Using workload analysis you can set different response time thresholds for each application based on required service levels. For simplicity, we set a 3 second critical threshold for all workloads.

We created a situation to generate a warning alert when WebSphere is serving static pages; refer to 5.4.6, “Example 8 - Static pages serving” on page 207 for details.

Based on our understanding of the PRR applicatio, we created situations to proactively monitor key metrics in WebSphere MQ and CICS. Refer to 5.4.3, “Example 4 - Identify a CICS TS response time problem” on page 188 and 5.4.5, “Example 7 - Transaction hang or time-out” on page 200 for details.

5.4 Analyzing the ITSO examples

5.4.1 Example 1 - Identify a DB2 delay in the application path

A specific user is experiencing slow response while other users work well.

In this example:

- ▶ We receive an alert on the navigator tree that a single invocation of a transaction has exceeded the predetermined response time threshold.
- ▶ Using the Longest Running Workloads workspace we see the parameters that the user specified to invoke this long-running transaction.
- ▶ We compare this invocation with the average response for the same workload on the All Workloads workspace. We determine that long response is limited to this one invocation.
- ▶ We drill down to the Selected Workloads Delays workspace to understand typical delays for this workload.
- ▶ To further diagnose this problem, we run an application trace for this transaction using the same input parameters and discover that a single SQL query is returning thousands of rows of data for this user.

Procedure

1. We receive an alert on the ebusiness navigator tree. Position the mouse pointer over the alert (red triangle icon) to show details. Figure 5-8 on page 174 shows that the Inventory Control transaction has a critical alert for server instance OMESRVA. Alert WAS_LongRun_Resp_Critical indicates that one or more workloads have exceeded the response time threshold for a single invocation.

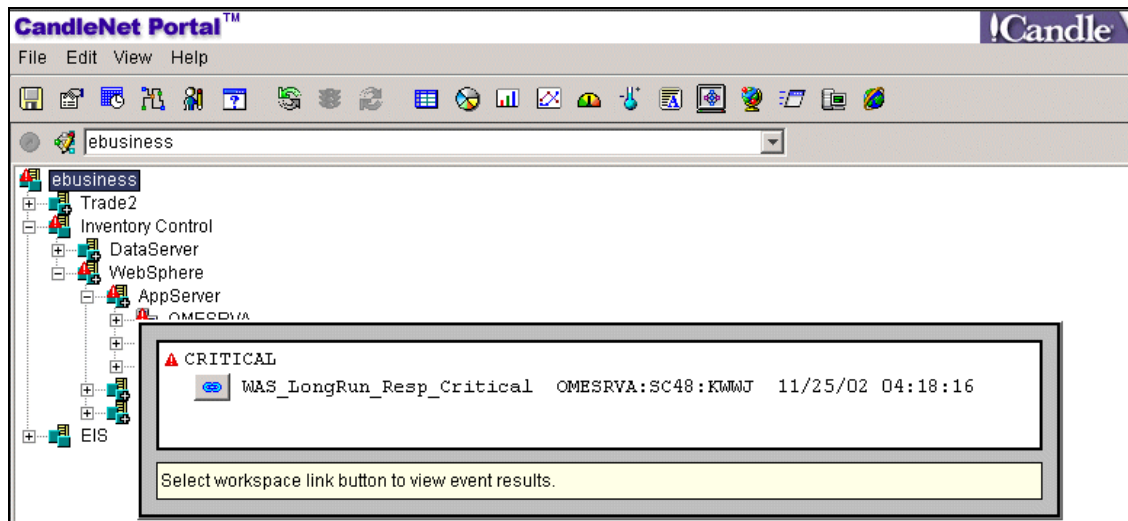


Figure 5-8 Alert WAS_LongRun_Resp_Critical on the ebusiness navigator tree

- Click the link button for `WAS_LongRun_Resp_Critical` in the critical alert window to display the Current Situation Values table. Figure 5-9 displays one servlet and three EJB methods that have exceeded the response time threshold. For example, the servlet has a response time of over 9 seconds (9307 ms).

Response Time	Server Name	Workload Type	Class Name	Method Name
9307	OMESRVA	Servlet	_OSResults_jsp_2	_jspService(warehouseId=26,custom...
9214	OMESRVA	EJB Method	orderStatusSessionPackage.OrderSt...	orderStatusSession("orderStatusSes...
8520	OMESRVA	EJB Method	customerEntityPackage.EJSBMP Cust...	findCustomerByLastName("MIN",26,2...
8512	OMESRVA	EJB Method	customerEntityPackage.CustomerEnt...	ejbFindCustomerByLastName("MIN",...

Figure 5-9 Current situation values

Clicking the link button in Figure 5-8 on page 174 also expands the navigator tree to display the Longest Running Workloads selection for instance OMESRVA.

- Select the Longest Running Workloads workspace for server instance OMESRVA.

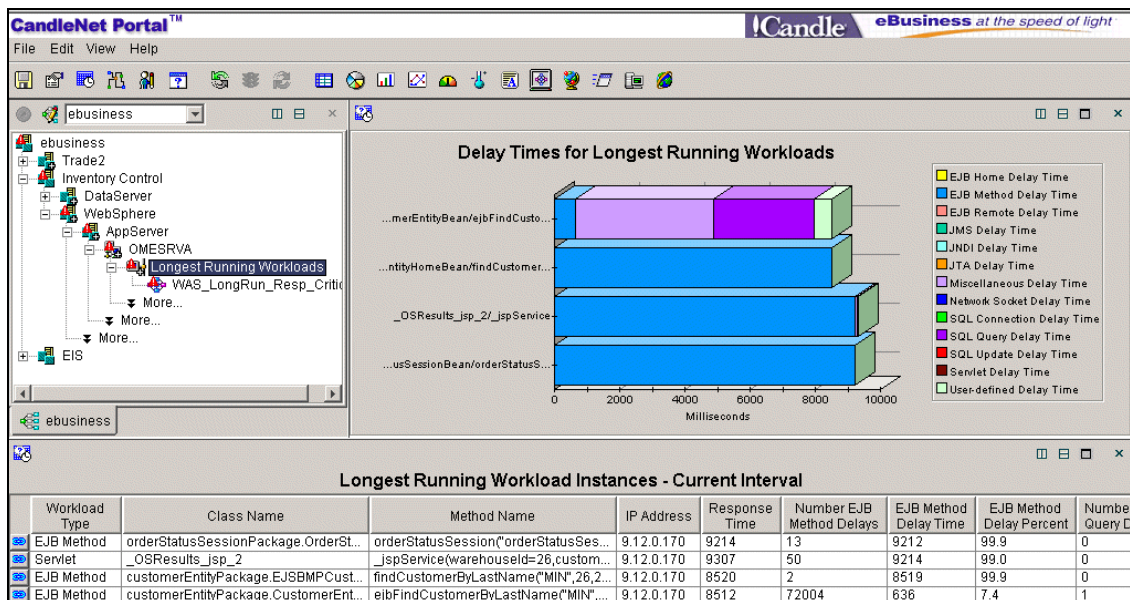


Figure 5-10 Longest Running Workloads workspace

The bar chart view in Figure 5-10 shows that four workloads have exceeded the response time threshold for an individual invocation. In this example, the response times are greater than 8 seconds (8000 ms).

We use the bar chart legend to analyze what these workloads are doing:

- Three workloads are spending most of their time waiting for other EJB methods (blue on the bar chart). We can't tell anything from these workloads. We need to focus on the downstream methods that are doing useful work.
- Response time for the fourth workload is composed of the following delays:
 - Calls to other EJB Methods Blue on the bar chart
 - Miscellaneous delays Lilac
 - SQL Queries Purple
 - User-defined delays Aqua

We want to discover if there is something unique about this invocation of this function. We start by determining the parameters specified by the user.

4. Looking more closely at the table view for the Longest Running Workloads workspace in Figure 5-10 on page 175, we sort the table to display the longest response times first.

Method Name
_jspService(warehouseId=26,customerId=12,CMPBMP=false,customerLastName=MIN,districtId=26,command=Manual)
orderStatusSession("orderStatusSessionPackage.OrderStatusInput@46018478")
findCustomerByLastName("MIN",26,26,false)
ejbFindCustomerByLastName("MIN",26,26,false)

Figure 5-11 Longest Running Workloads Method Names and parameters

In Figure 5-11 we widen the Method Name field to see the parameters that were input to the servlet and passed to `orderStatusSession` and the other methods:

- warehouseId=26
- customerId=12
- CMPBMP=false
- customerLastName=MIN
- districtId=26
- command=Manual

We know from Figure 5-10 on page 175 that three of the workloads are waiting for other EJB methods. These will not help us understand why the response time is slow. We want to examine the method that is performing the business logic. We deduce that `ejbFindCustomerByLastName` is the method that is performing the business logic. We want to understand where this method is spending its time. We see the following response time delays for

method `ejbFindCustomerByLastName` in the Longest Running Workload Instances table in Figure 5-10 on page 175:

Method Name	Number EJB Method Delays	EJB Method Delay Time	EJB Method Delay Percent	Number SQL Query Delays	SQL Query Delay Time	SQL Query Delay Percent
<code>ejbFindCustomerByLastName("MIN",...</code>	72004	636	7.4	1	3073	36.1

Figure 5-12 Long Running: EJB Method Delays and SQL Query Delays

Method Name	Number User-defined Delays	User-defined Delay Time	User-defined Delay Percent	Number Miscellaneous Delays	Miscellaneous Delay Time	Miscellaneous Delay Percent
<code>ejbFindCustomerByLastName("MIN",...</code>	144000	557	6.5	2	4246	49.8

Figure 5-13 Long Running: User Defined Delays and Miscellaneous Delays

From Figure 5-12 and Figure 5-13 we see that the response time for method `ejbFindCustomerByLastName` is broken into the following delay categories:

EJB Method calls 636 ms
 SQL Queries 3073 ms
 User defined delays 557 ms
 Miscellaneous delays 4246 ms

We also see that the number of SQL Query Delays is one, so the 3-second delay for SQL queries is attributed to one SQL call. This single execution of `ejbFindCustomerByLastName` is making 72004 EJB method calls, and 14400 calls to a user-defined method. This transaction is doing a lot of processing for a single query.

We want to understand how these response times compare with a typical execution of method `ejbFindCustomerByLastName`. If all invocations of this method have a long response time, then we may have a WebSphere or system problem. If this specific invocation is the only one with long response, then we need to understand what is unique about this invocation.

5. Select All Workloads from the `ebusiness` navigator tree.

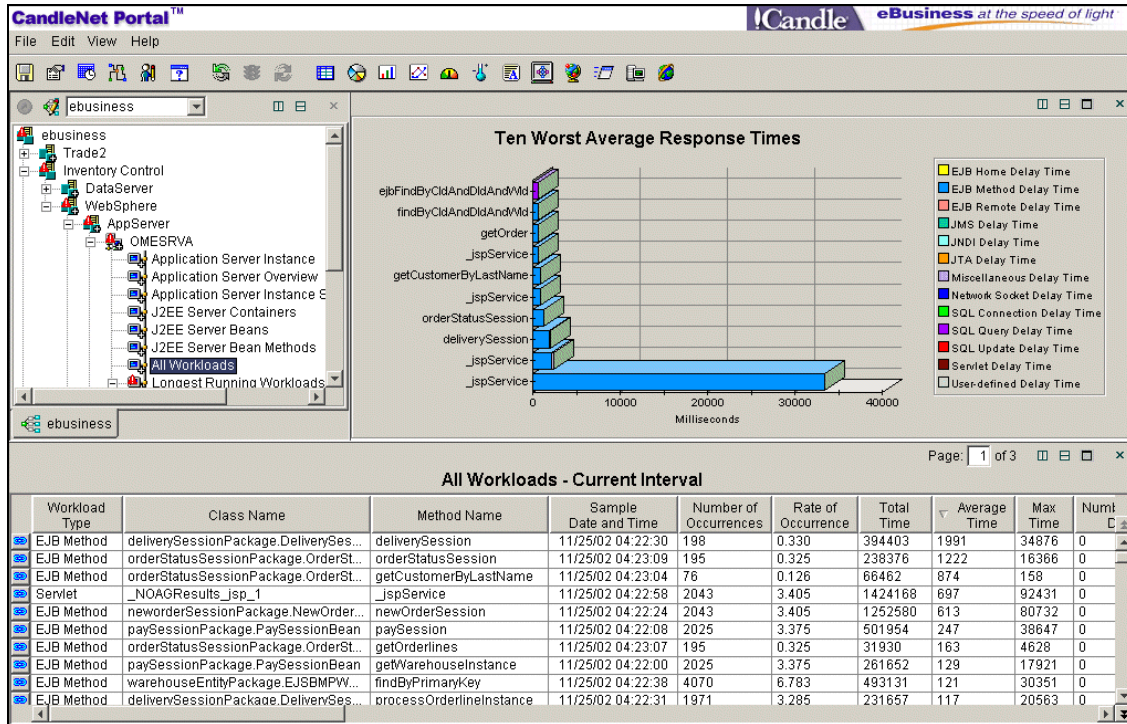


Figure 5-14 All Workloads workspace

Looking more closely at the table view from the All Workloads workspace in Figure 5-14, we filter by Method Name=ejbFindCustomerByLastName. Then we see the response time categories for an average invocation of `ejbFindCustomerByLastName`:

Method Name	Average Time	Number EJB Method Delays	EJB Method Delay Time	EJB Method Delay Percent	Number SQL Query Delays	SQL Query Delay Time	SQL Query Delay Percent
ejbFindCustomerByLastName	60	5239	19	32.5	397	15	25.9

Figure 5-15 Average: EJB Method Delays and SQL Query Delays

Method Name	Average Time	Number User-defined Delays	User-defined Delay Time	User-defined Delay Percent	Number Miscellaneous Delays	Miscellaneous Delay Time	Miscellaneous Delay Percent
ejbFindCustomerByLastName	60	7302	2	3.7	697	22	37.7

Figure 5-16 Average: User Defined Delays and Miscellaneous Delays

In Figure 5-15 and Figure 5-16 the average response time for method `ejbFindCustomerByLastName` is 60 ms. We compare this with the 8-second response we saw for one invocation of `ejbFindCustomerByLastName` in

Figure 5-10 on page 175. We deduce that the problem is limited to this one long-running invocation.

The average response time is broken down as follows:

EJB Method calls 19 ms
 SQL Queries 15 ms
 User defined delays 2 ms
 Miscellaneous delays 22 ms

We also see that the typical execution in Figure 5-15 on page 178 makes 5239 EJB method calls compared to 72004 for our example in Figure 5-12 on page 177.

We want a better understanding of the typical execution to see if we can deduce why our example is performing so poorly.

- Click the link button for method `ejbFindCustomerByLastName` on the All Workloads table view to navigate to the Selected Workload Delays workspace.

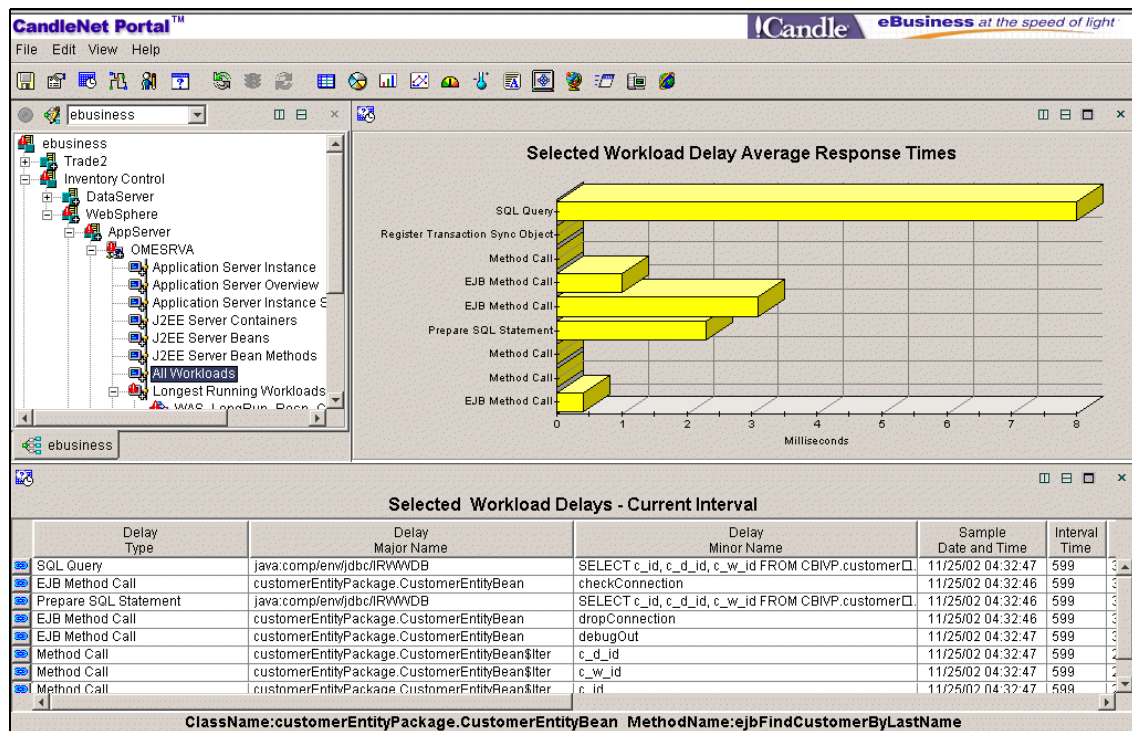


Figure 5-17 Selected Workload Delays workspace

The table view for the Selected Workload Delays workspace in Figure 5-18 displays the functions that are performed by an typical execution of method `ejbFindCustomerByLastName`.

We see a Prepare SQL Statement and SQL Query for a SELECT statement to datasource IRWWDB. We also see calls to user-defined methods from the `customerEntityPackage`. We can pass this information to the application support team or database administrator to start problem diagnosis.

Delay Type	Delay Major Name	Delay Minor Name
SQL Query	java.comp.enw.jdbc.IRWWDB	SELECT c_id, c_d_id, c_w_id FROM CBIVP.customer□
EJB Method Call	customerEntityPackage.CustomerEntityBean	checkConnection
Prepare SQL Statement	java.comp.enw.jdbc.IRWWDB	SELECT c_id, c_d_id, c_w_id FROM CBIVP.customer□
EJB Method Call	customerEntityPackage.CustomerEntityBean	dropConnection
EJB Method Call	customerEntityPackage.CustomerEntityBean	debugOut
Method Call	customerEntityPackage.CustomerEntityBean\$Iter	c_d_id
Method Call	customerEntityPackage.CustomerEntityBean\$Iter	c_w_id
Method Call	customerEntityPackage.CustomerEntityBean\$Iter	c_id

Figure 5-18 Selected Workload Delays table view

From Figure 5-11 on page 176 we can also provide the parameters that were input to the servlet that resulted in the long response:

- warehouseId=26
- customerId=12
- CMPBMP=false
- customerLastName=MIN
- districtId=26
- command=Manual

Detailed problem analysis

In order to understand why specific parameters cause a long response time for the `orderStatus` transaction, we trace a second execution of this transaction with the same set of input parameters.

Depending on the business function performed by a transaction, it may not be possible to run the transaction again without impacting the integrity of the data. For example, you cannot repeat a transaction that removes money from a bank account. In that case, the application support team can run the application trace in a non-production environment with a copy of the database.

We know from Figure 5-17 on page 179 that the `orderStatus` transaction does an SQL query, but does not perform any SQL updates. We determine that it is safe in our environment to invoke the `orderStatus` transaction with the same set of input parameters.

1. Dynamically start an application trace using the Take Action command WAS390 Start Application Tracing, then submit the orderStatus transaction with the same parameters.
2. Select **Application Trace** from the navigator tree, and link to the trace.

Method Name	Event Type	Object	Parameters
executeQuery	Method Entry	COM.ibm.db2os390.sqlj.jdbc.DB2SQLJPreparedStatement...	java.comp/enwjdbc/IRWWDB/SELECT c_id, c
executeQuery	Method Exit	COM.ibm.db2os390.sqlj.jdbc.DB2SQLJResultSet@439fb40d	
next	Method Entry	COM.ibm.db2os390.sqlj.jdbc.DB2SQLJResultSet@439fb40d	java.comp/enwjdbc/IRWWDB/SELECT c_id, c
next	Method Exit		
next	Method Entry	COM.ibm.db2os390.sqlj.jdbc.DB2SQLJResultSet@439fb40d	java.comp/enwjdbc/IRWWDB/SELECT c_id, c
next	Method Exit		
next	Method Entry	COM.ibm.db2os390.sqlj.jdbc.DB2SQLJResultSet@439fb40d	java.comp/enwjdbc/IRWWDB/SELECT c_id, c
next	Method Exit		
next	Method Entry	COM.ibm.db2os390.sqlj.jdbc.DB2SQLJResultSet@439fb40d	java.comp/enwjdbc/IRWWDB/SELECT c_id, c
next	Method Exit		

Figure 5-19 Application Trace

Figure 5-19 contains an extract from the application trace, showing calls from method `ejbFindCustomerByLastName`. We see that this transaction calls method `executeQuery` with a SQL SELECT statement, then repeatedly calls method `next` to process the next row in the result set. There are thousands of calls to method `next` in the complete trace.

We determine that the long response time on this invocation of `orderStatus` is due to the large amount of data being returned by DB2. At this point we turn the problem over to the application support team to determine if this condition is expected.

5.4.2 Example 3 - Detect a memory leak

There is a memory leak in one of the applications. In this example:

1. We receive an alert on the ebusiness navigator tree.

Position the mouse pointer over the alert (red triangle icon) to show details. Figure 5-20 on page 182 shows that the Inventory Control transaction has a critical alert for server instance OMESRVA. The alert `WAS_Workload_AvgResp_Critical` indicates that one or more workloads have exceeded the threshold for average response time.

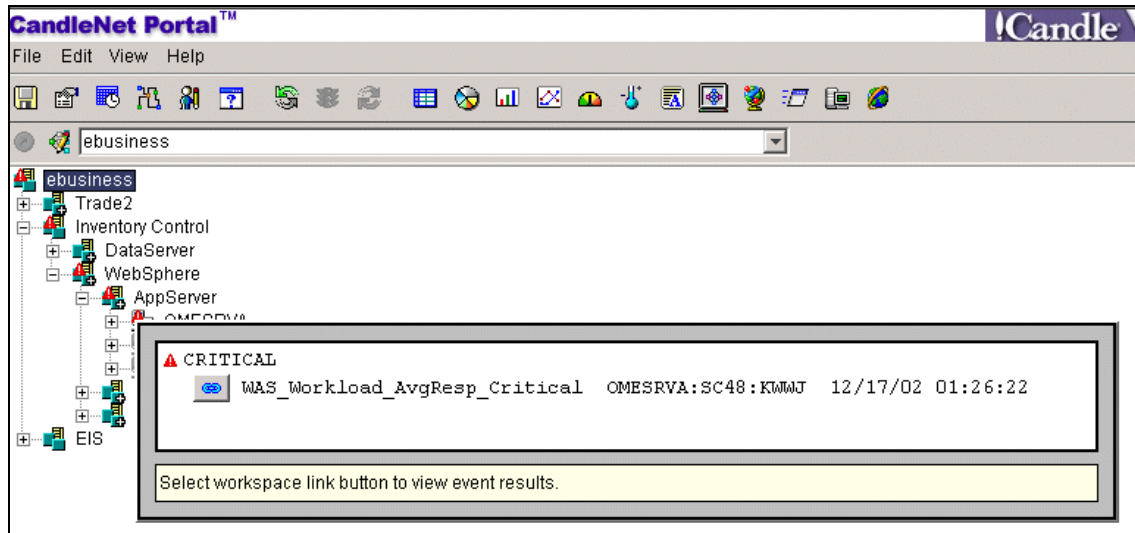


Figure 5-20 Alert on the navigator tree view

2. Click the link button for `WAS_Workload_AvgResp_Critical` in the critical alert window to display the Current Situation Values table. Figure 5-21 displays one servlet (method name `_JSPService`) and one EJB Method (`deliverySession`) that have exceeded the response time threshold. For example, the servlet has an average response of over 4 seconds (4459 ms).

Average Time	Server Name	Workload Type	Class Name	Method Name	Number of Occurrences
4459	OMESRVA	Servlet	<code>_DEAGResults_jsp_3</code>	<code>_jspService</code>	24
3533	OMESRVA	EJB Method	<code>deliverySessionPackage.DeliverySessionBe...</code>	<code>deliverySession</code>	25

Figure 5-21 Current situation values

Clicking on the link button in Figure 5-20 also expands the navigator tree to display the All Workloads selection for instance OMESRVA.

3. Select the All Workloads workspace for instance OMESRVA.

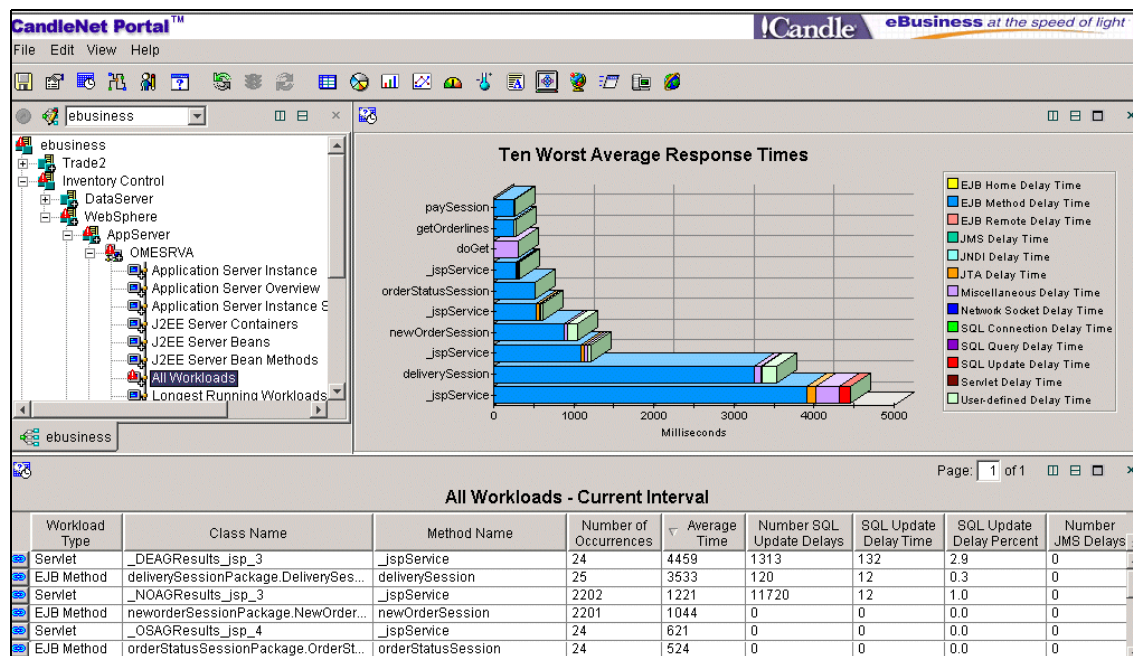


Figure 5-22 All Workloads workspace

The bar chart view in Figure 5-22 shows that the workload with the worst average response time is a servlet (class name `_DEAGResults_jsp_3`, method name `_jspService`). This servlet is spending most of its time waiting for EJB Method calls (blue on the bar chart). Reviewing the other workloads in the bar chart view, there is no significant time waiting for resources such as JMS, SQL, or user-defined resources.

We want to understand if the long response time for this servlet is a one-time occurrence or a persistent problem, so we review the average response time for this workload over the last hour.

4. Either click the link button on the All Workloads table row for method `_jspService`, or right-click on the bar representing `_jspService` in the All Workloads bar chart view to navigate to the Selected Workload - History workspace:

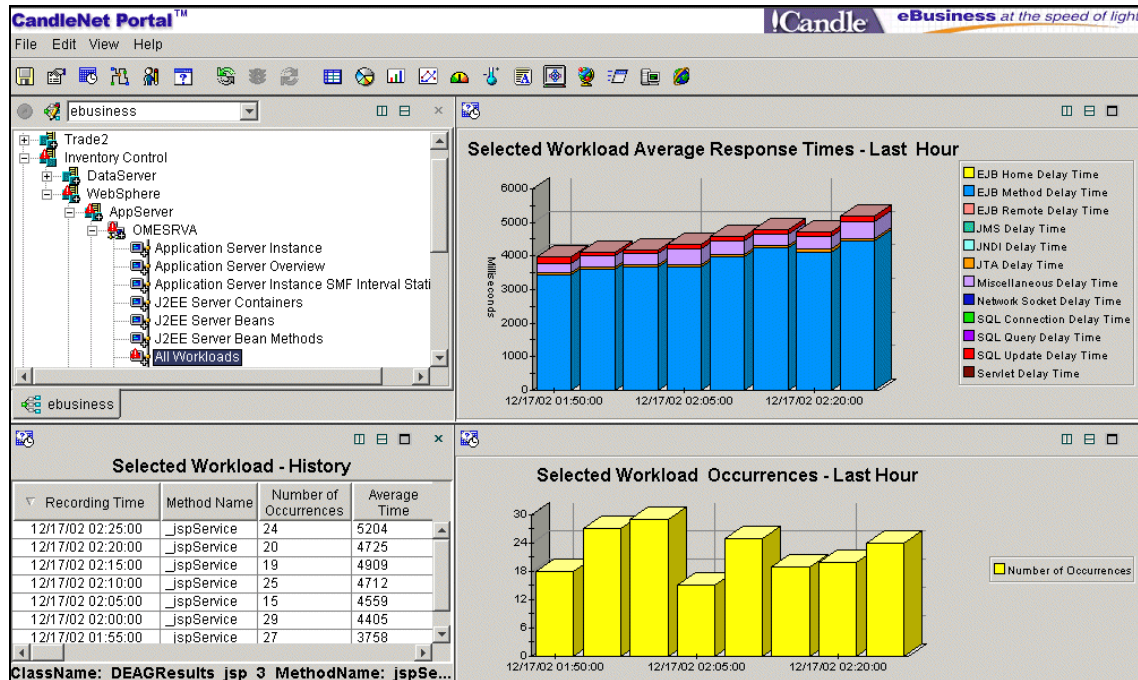


Figure 5-23 Selected Workload - History workspace

Figure 5-23 contains two bar chart views:

- The Selected Workload Average Response Times view shows that the average response time for this workload has been increasing over the last hour.
- The Selected Workload Occurrences view shows that the number of occurrences for this workload has varied over the last hour, but the overall trend is slightly decreasing.

One possible explanation is that the overall throughput for this server instance has increased causing the average response time for this workload to increase.

5. Select **HTTP Sessions** from the navigator tree view. Figure 5-24 on page 185 shows that the number of HTTP sessions has decreased over the last hour. We know that the number of HTTP sessions has decreased, yet the average response times have increased over the last hour. Also, we do not see any alerts on the navigator tree for the connected systems, so we do not appear to have a system problem. One possible explanation is that we are running low on memory, causing the application server instance to spend more time on garbage collections, at the expense of transaction throughput and response times.

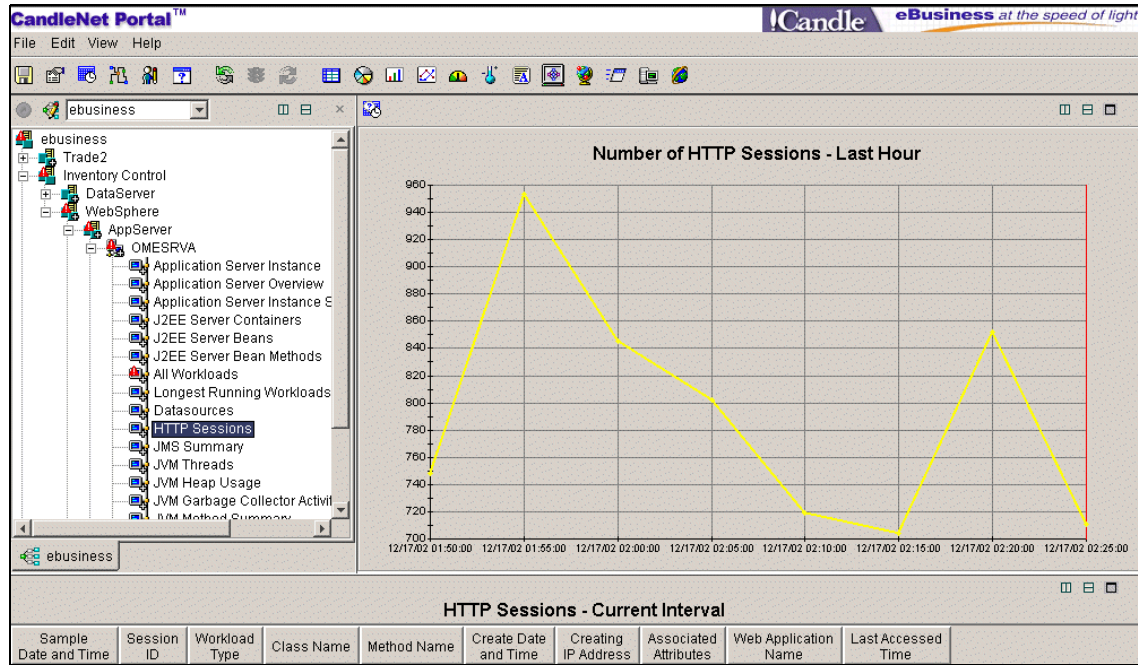


Figure 5-24 HTTP Sessions workspace

6. Select **Application Server Overview** on the navigator tree view. The Application Server Overview table view in Figure 5-25 on page 186 shows current JVM memory usage for the two server regions for this instance. We plot the JVM memory use over the last hour for one of the server regions (it doesn't matter which). We see that the memory use has increased significantly over the last hour.

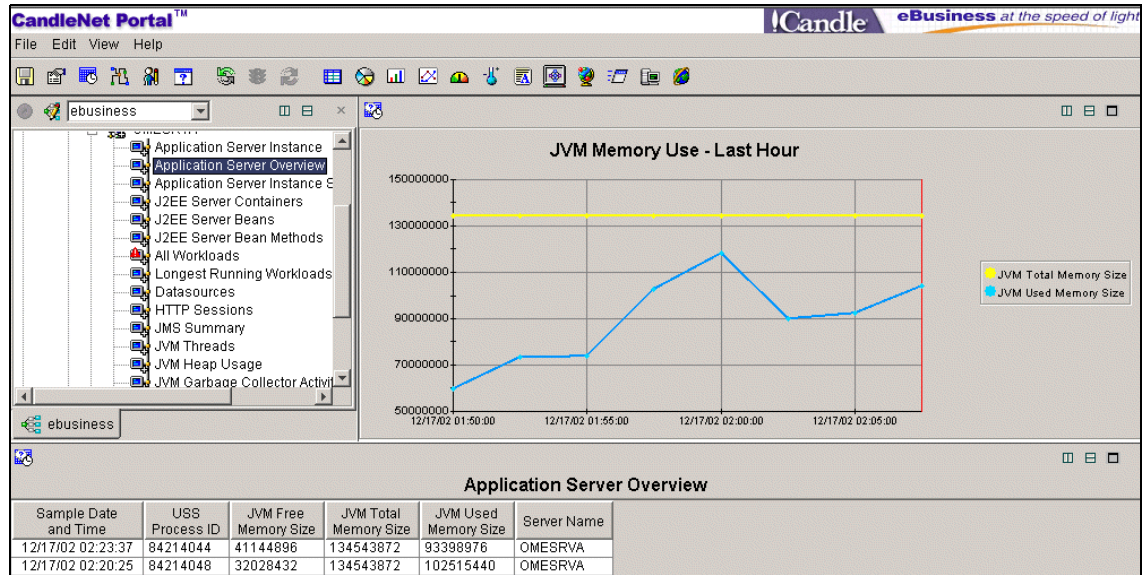


Figure 5-25 Application Server Overview workspace

This workspace samples the JVM memory use at regular intervals, but it does not show when the garbage collector is running, so it cannot show the minimum memory used after each garbage collection. However, the overall increase in memory use in conjunction with the increased response times and decreased throughput are sufficient indicators to justify enabling the JVM profiler interface and recycling the server region.

- Let the application server run for a while, then select **JVM Garbage Collections** on the navigator tree.

The Garbage Collections / Interval view on the JVM Garbage Collections workspace from Figure 5-26 on page 187 shows that the number of garbage collections has increased steadily over the last hour, and has increased exponentially in the last ten minutes.

The Bytes Free After Garbage Collection view on the JVM Garbage Collections workspace shows that the amount of free storage has decreased linearly over the last hour, from 80 MB to 10 MB.

The %CPU Used view on the JVM Garbage Collections workspace shows that CPU usage by the garbage collector has increased non-linearly over the last hour. The garbage collector is currently using about 17% of the CPU.

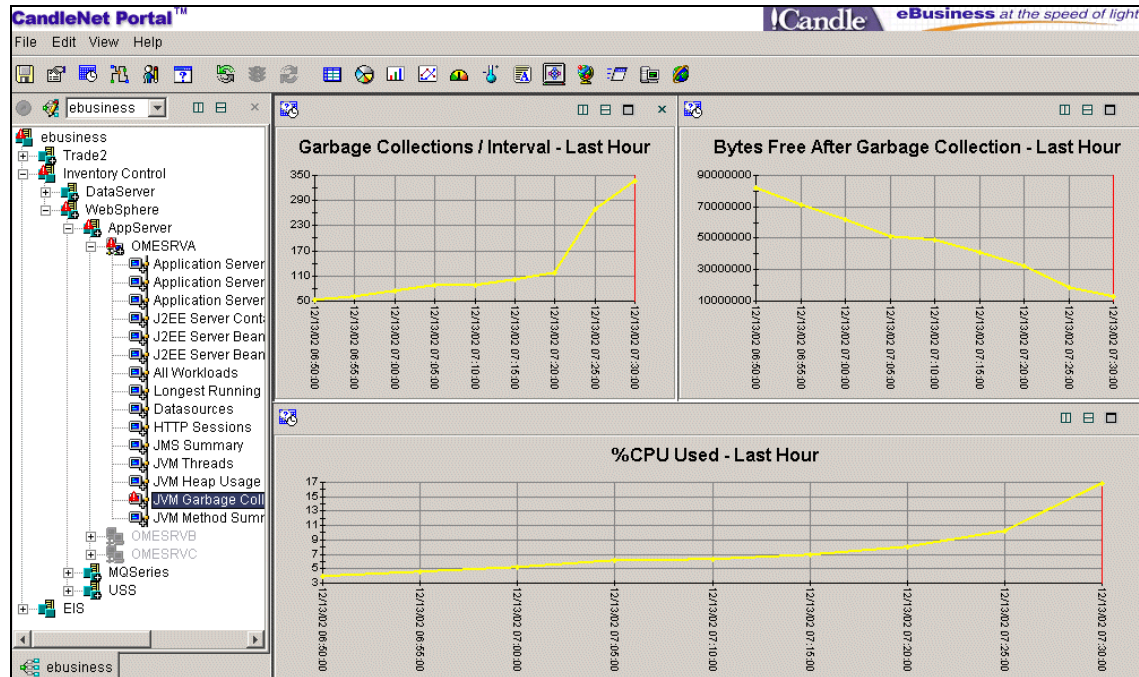


Figure 5-26 JVM Garbage Collections workspace

It appears that we have a memory leak. The garbage collector is running more and more frequently, yet the JVM Heap is running out of memory.

8. Select **JVM Heap Usage** on the navigator tree view. The Ten Highest Heap Size view in Figure 5-27 on page 188 shows that one class, `weberwno/NOController$NewOrderData`, is using 160 MB of heap memory (168,245,616 bytes). This class is the likely cause of the memory leak problem.

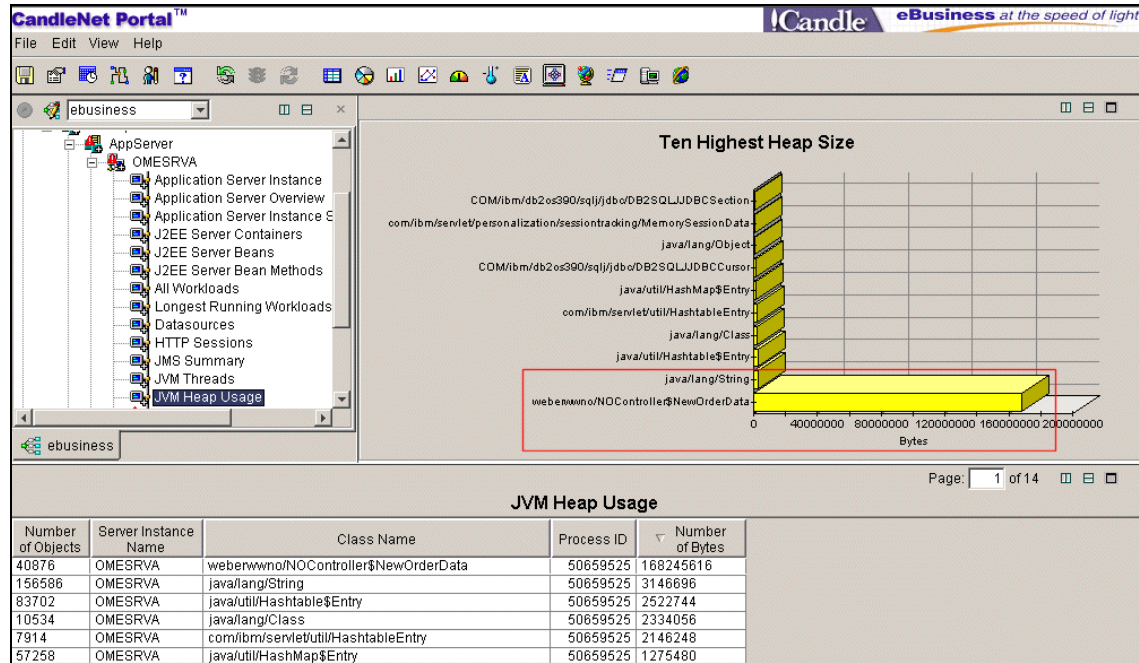


Figure 5-27 JVM Heap Usage workspace

5.4.3 Example 4 - Identify a CICS TS response time problem

There is a problem in CICS Transaction Server that is impacting WebSphere.

In this example:

- ▶ We receive an alert on the ebusiness navigator tree that the average response time for some workloads has exceeded the predetermined threshold.
- ▶ We also receive an alert on the ebusiness navigator tree indicating a problem in CICS. We need to determine if the two issues are related.
- ▶ Using the All Workloads workspace, we see that one workload has a long response time, possibly calling the CICS Transaction Gateway (CICS TG).
- ▶ We drill down to the Selected Workload Delays workspace and confirm that the delay is due to a CICS TG request.
- ▶ Using OMEGAMON XE for CICSplex, we see that a particular transaction class has met its limit, causing CICS tasks for transaction CSMI to wait for first dispatch. This confirms that workloads on WebSphere using CICS TG are being impacted by CICS.

Procedure

1. We receive an alert on the ebusiness navigator tree. Position the mouse pointer over the alert (red triangle icon) to show details. Figure 5-28 shows that the EIS application has two critical alerts:
 - a. Alert `WAS_Workload_AvgResp_Critical` for server instance `OMTSRVA` indicates that one or more workloads have exceeded the threshold for average response time.
 - b. Alert `CICSplex_ClassMax_Critical` for CICS region `SCSCERW1` indicates that a CICS transaction class has exceeded its limit.

We first focus on the response time alert for WebSphere to see if it is related to the CICS alert.

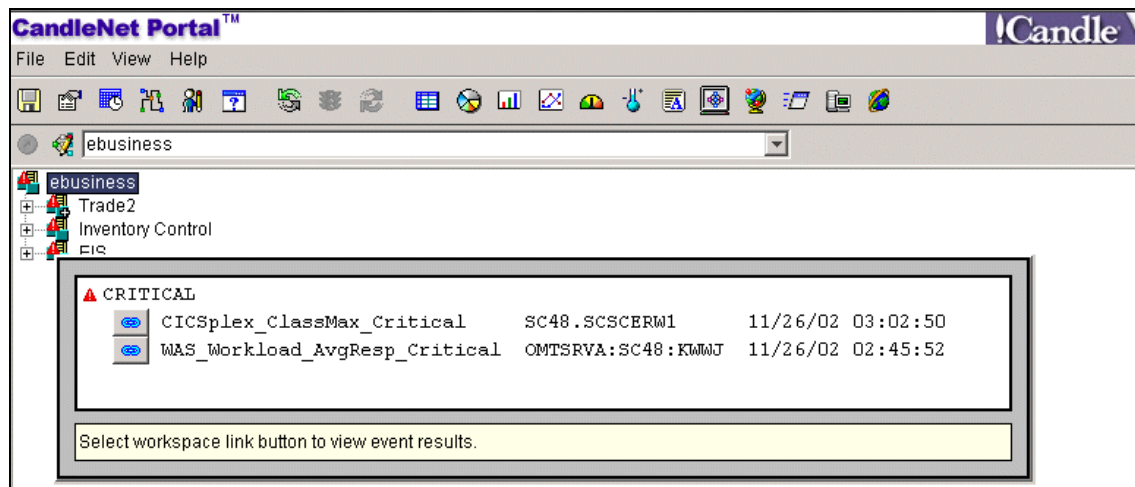


Figure 5-28 Two alerts on the ebusiness navigator tree

2. Click the link button for `WAS_Workload_AvgResp_Critical` in the critical alert window to display the Current Situation Values table. Figure 5-29 displays one servlet and an EJB Method that have exceeded the response time threshold. For example, the servlet has an average response of over 5 seconds (5658 ms).

Average Time	Server Name	Inst	Workload Type	Class Name	Method Name
5658	OMTSRVA		Servlet	_JMSResults_jsp_3	_jspService
5535	OMTSRVA		EJB Method	erwwcics.ctg.pc.ERWWCTGPCBean	priceChangeEJBdriver

Figure 5-29 Current situation values

Clicking the link button for `WAS_Workload_AvgResp_Critical` in Figure 5-27 on page 188 also expands the navigator tree to display the All Workloads selection for server instance OMTSRVA.

3. Select the All Workloads workspace for server instance OMTSRVA.

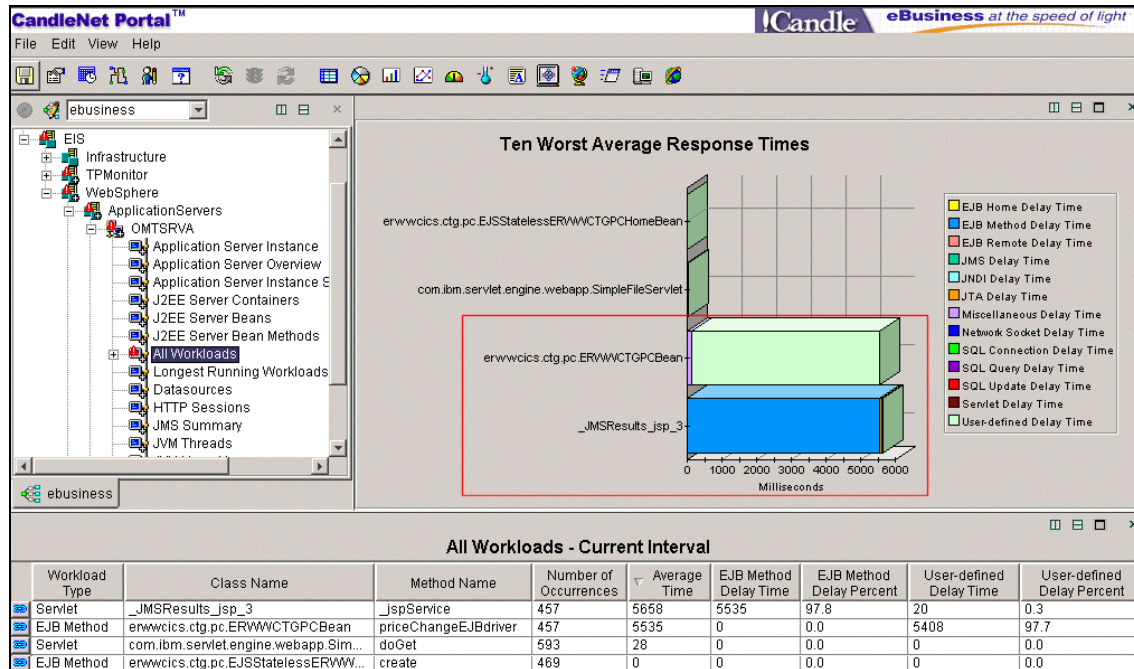


Figure 5-30 All Workload workspaces

The bar chart view in Figure 5-30 shows two workloads with average response times of over 5 seconds (5000 ms). Workload `_JMSResults_jsp_3` has the longest response time but it is spending most of its time waiting for EJB Method calls (blue on the bar chart), so it cannot help us determine the cause of the delay. Workload `erwwcics.ctg.pc.ERWWCTGPCBean` is spending almost 5.5 seconds on a user-defined wait (aqua on the bar chart).

This workload contains `ctg` in its name, so we suspect that this class is related to CICS TG calls. Also, in the current PathWAI release, method calls for CICS TG are instrumented as user-defined delays. We need to drill down to see the actual delays to determine if this workload is really waiting for CICS TG requests.

Note: In a future PathWAI release CICS delays will be displayed as a separate wait category in the All Workloads bar chart view.

- Right-click the bar for `erwwcics.ctg.pc.ERWWCTGPCBean` in Figure 5-30 on page 190 and navigate to the Selected Workload Delays workspace.

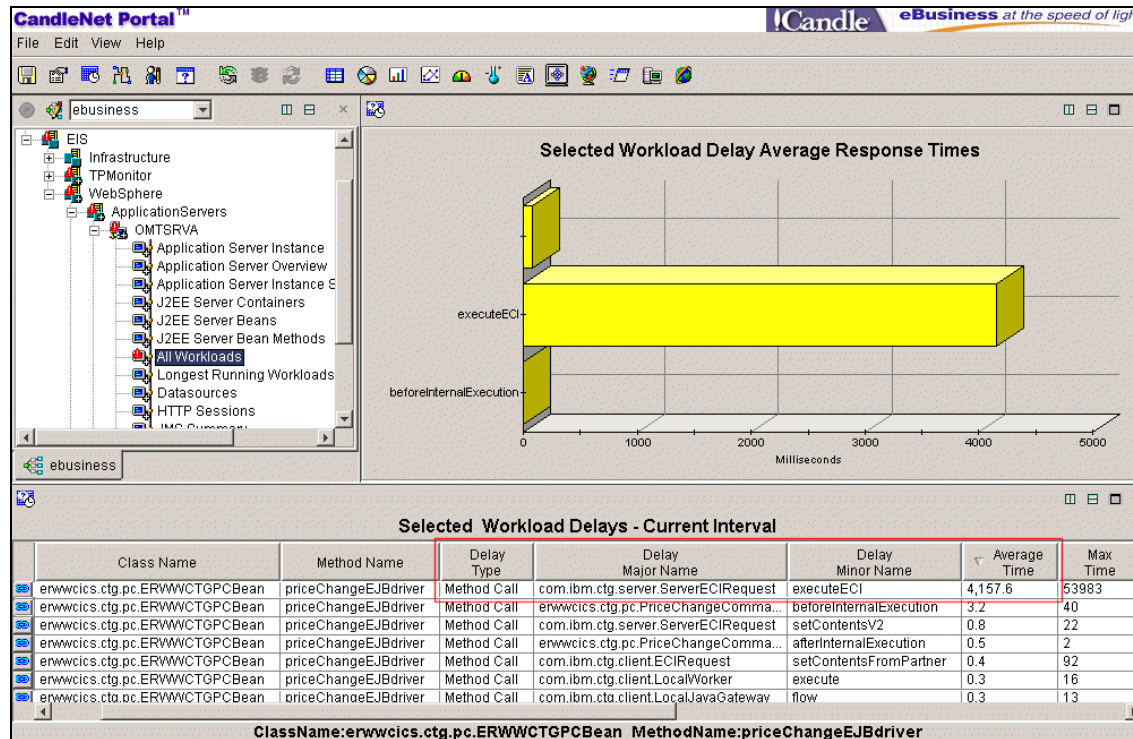


Figure 5-31 Selected Workload Delays workspace

Figure 5-31 shows detailed delay information for class: `erwwcics.ctg.pc.ERWWCTGPCBean`, method: `priceChangeEJBdriver`.

We sort the table view in descending order by Average Time to find the longest delay for method `priceChangeEJBdriver`. The largest delay is method call `ExecuteECI` for class `com.ibm.ctg.server.ServerECIRequest`. This is the IBM-provided class for CICS TG requests. This request has an average response time of over 4 seconds (4,157.6 ms).

It appears that WebSphere workloads are being delayed due to CICS. We use OMEGAMON XE for CICSplex to investigate the critical alert for CICS.

- Click the link button for alert `CICSplex_ClassMax_Critical` in Figure 5-28 on page 189 to display the Current Situation Values table for this alert.

Percent of Limit	Origin Node	System ID	CICS Region Name	Class Name	Class Limit	Times at Limit	Current Tasks
1200	SC48.SCSCERW1	SC48	SCSCERW1	CSMICLAS	1	1	12

Figure 5-32 CICS Current situation values

Figure 5-32 shows that CICS transaction class CSMICLAS is at 1200% of its class limit. Clicking the link button for CICSplex_ClassMax_Critical in Figure 5-28 on page 189 also expands the navigator tree to display the Task Class Analysis selection for CICS region SCSCERW1.

6. Select the Task Class Analysis workspace for CICS region SCSCERW1.

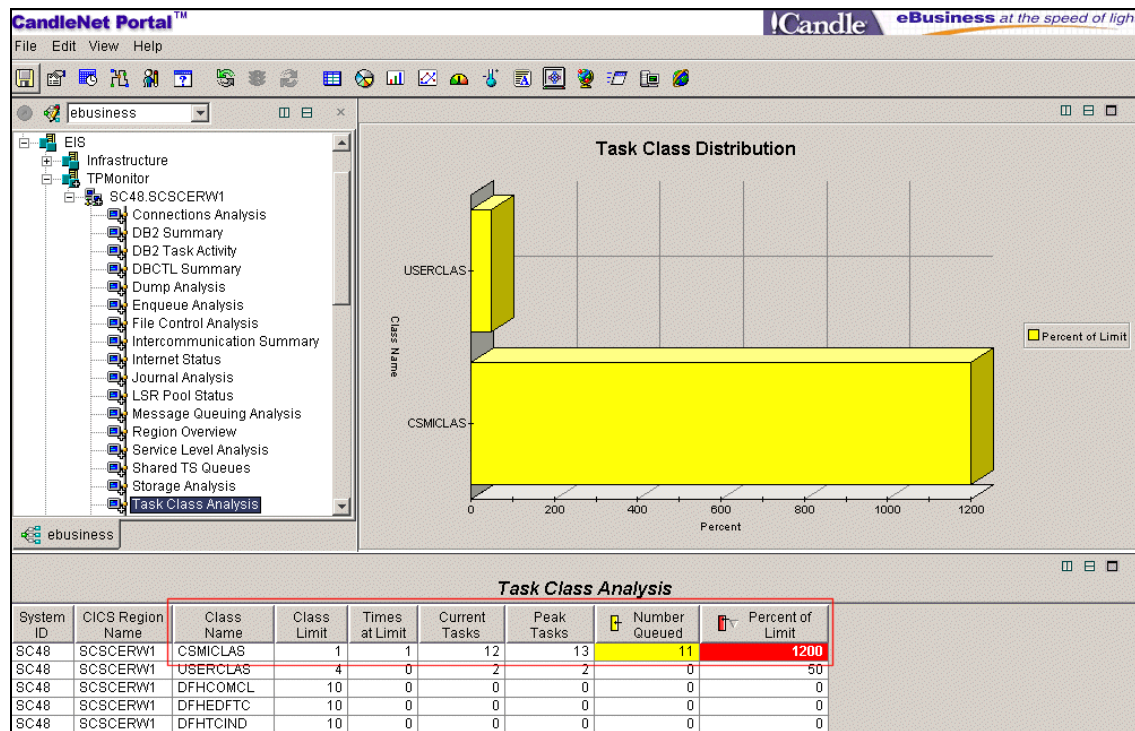


Figure 5-33 Task Class Analysis workspace

The table view in Figure 5-33 confirms that transaction class CSMICLAS has a limit of one task and there are currently 11 tasks queued.

CSMI tasks processing CICS TG requests are queued waiting for first dispatch, resulting in response time delays in WebSphere.

Example 4 - Advanced procedure

An alternative to the previous procedure is to use the custom business view for the PRR application; refer to “Customized business view” on page 172. With this approach we can proactively monitor all the key components of the PRR application from one workspace.

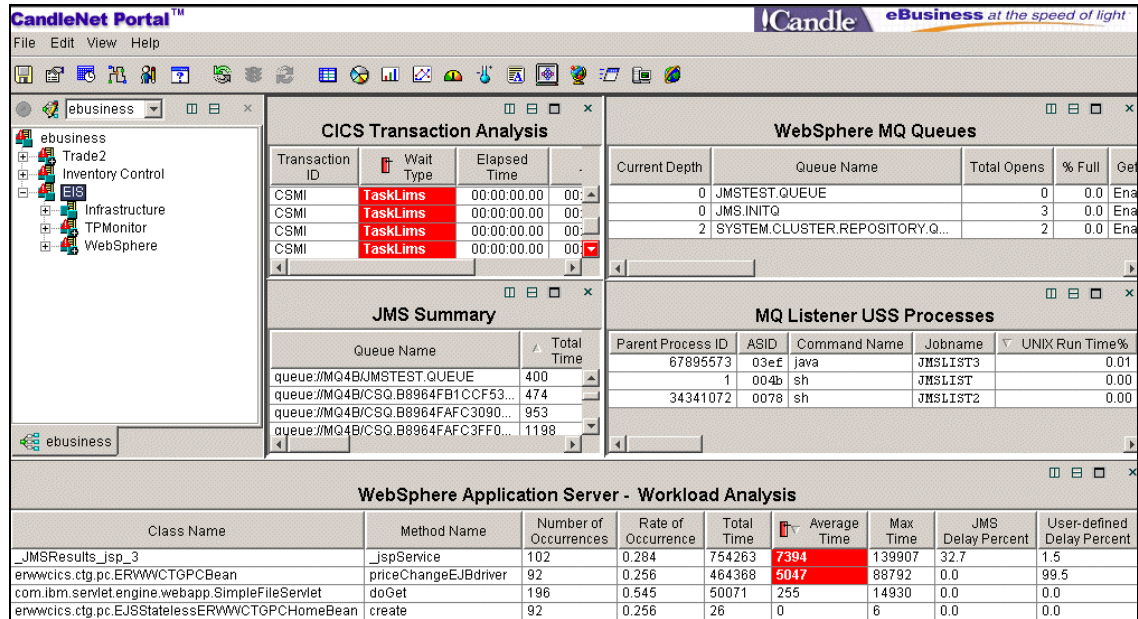


Figure 5-34 Custom business view

Figure 5-34 shows the custom business view that we configured for application PRR. It uses OMEGAMON DE to integrate the following information into a single workspace:

- The CICS Transaction Analysis view displays task information for CICS region SCSCERW1. This region processes the CICS TG requests for application EIS. Product-provided situations automatically raise an alert for CICS task-related issues.
- The WebSphere Application Server - Workload Analysis view displays average response times for workloads on server instance OMTSRVA. We configured this view to alert us when the average response time for a workload is greater than 3 seconds.

The custom view also includes WebSphere MQ Queues, JMS Summary, and MQ Listener USS Processes views that are not pertinent to this example. These views are used by another business function within the PRR application; refer to “Example 7 - Advanced procedure” on page 205 for more details.

Looking more closely at the CICS Transaction Analysis view from Figure 5-34 on page 193, we see several CSMI transactions that are waiting due to a CICS transaction class limit:

Transaction ID	Wait Type	Elapsed Time
CSMI	TaskLims	00:00:00.00
CSMI	TaskLims	00:00:00.00
CSMI	TaskLims	00:00:00.00
CSMI	TaskLims	00:00:00.00

Figure 5-35 CICS Transaction Analysis view

From the WebSphere Application Server - Workload Analysis view in Figure 5-34 on page 193, we see that application EIS has two workloads that are experiencing average response times of over 5 seconds (7394 ms):

Class Name	Method Name	Number of Occurrences	Rate of Occurrence	Total Time	Average Time
_JMSResults_jsp_3	_jspService	102	0.284	754263	7394
erwwcics.ctg.pc.ERWWCTGPCBean	priceChangeEJBdriver	92	0.256	464368	5047
com.ibm.servlet.engine.webapp.SimpleFileServlet	doGet	196	0.545	50071	255
erwwcics.ctg.pc.EJSStatelessERWWCTGPCHomeBean	create	92	0.256	26	0

Figure 5-36 WebSphere Application Server - Workload Analysis view

In one workspace, the custom business view for application PRR shows long response times for server instance OMTSRVA and issues with CICS CSMI transactions that process CICS TG requests for this application.

5.4.4 Example 6 - Isolate a DB2 problem

Users complain that some of their business functions are very slow while other functions work well.

In this example:

- ▶ We receive an alert on the navigator tree that average response times for some workloads have exceeded the predetermined threshold.
- ▶ Using the All Workloads workspace we see that some methods are experiencing delays for SQL queries.
- ▶ We drill down to the Selected Workload Delays workspace to view the individual SQL delays for the poorly responding methods.
- ▶ We see that one specific SELECT statement has an excessive response time. We deduce that the problem is in DB2.

- ▶ We use OMEGAMON XE for DB2 to determine the cause of the delay in DB2.
- ▶ We select the **Detailed Thread Exception** workspace to see the DB2 threads used by our application server. We see that our application server is accessing a specific plan with high read I/O rates.
- ▶ We select the **Volume Activity** workspace to review physical I/O rates for the volumes used by DB2. We determine that DB2 is performing physical I/O rather than reading from the buffer pool. It appears that a specific SELECT statement from WebSphere is causing excessive physical I/O reads by DB2.

Procedure

1. We receive an alert on the ebusiness navigator tree. Position the mouse pointer over the alert (red triangle icon) to show details. Figure 5-37 shows that the Inventory Control application has a critical alert for server instance OMESRVB. The alert WAS_Workload_AvgResp_Critical indicates that one or more workloads have exceeded the threshold for average response time.

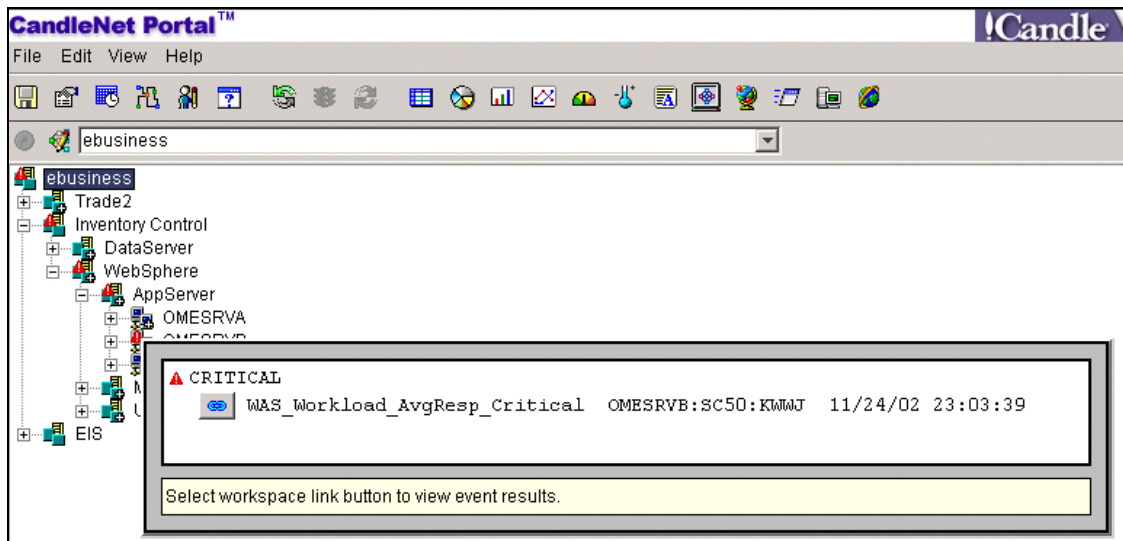


Figure 5-37 Alert WAS_Workload_AvgResp_Critical on the ebusiness navigator tree

2. Click the link button for WAS_Workload_AvgResp_Critical in the critical alert window to display the Current Situation Values table.

Average Time	Server Name	Workload Type	Class Name	Method Name
328630	OMESRVB	Servlet	_OSResults_jsp_0	_jspService
324116	OMESRVB	EJB Method	orderStatusSessionPackage.OrderStatusSessionBean	orderStatusSession
312997	OMESRVB	EJB Method	orderStatusSessionPackage.OrderStatusSessionBean	getCustomerInstance
312995	OMESRVB	EJB Method	customerEntityPackage.EJSBMPCustomerEntityHomeBean	findByPrimaryKey
158226	OMESRVB	EJB Method	customerEntityPackage.CustomerEntityBean	ejbFindByPrimaryKey

Figure 5-38 Current situation values

Figure 5-38 displays one servlet and four EJB Methods that have exceeded the response time threshold. For example, the servlet has an average response of over 320 seconds (328630 ms).

Clicking the link button in Figure 5-37 also expands the navigator tree to display the All Workloads selection for instance OMESRVB.

3. Select the All Workloads workspace for instance OMESRVB.

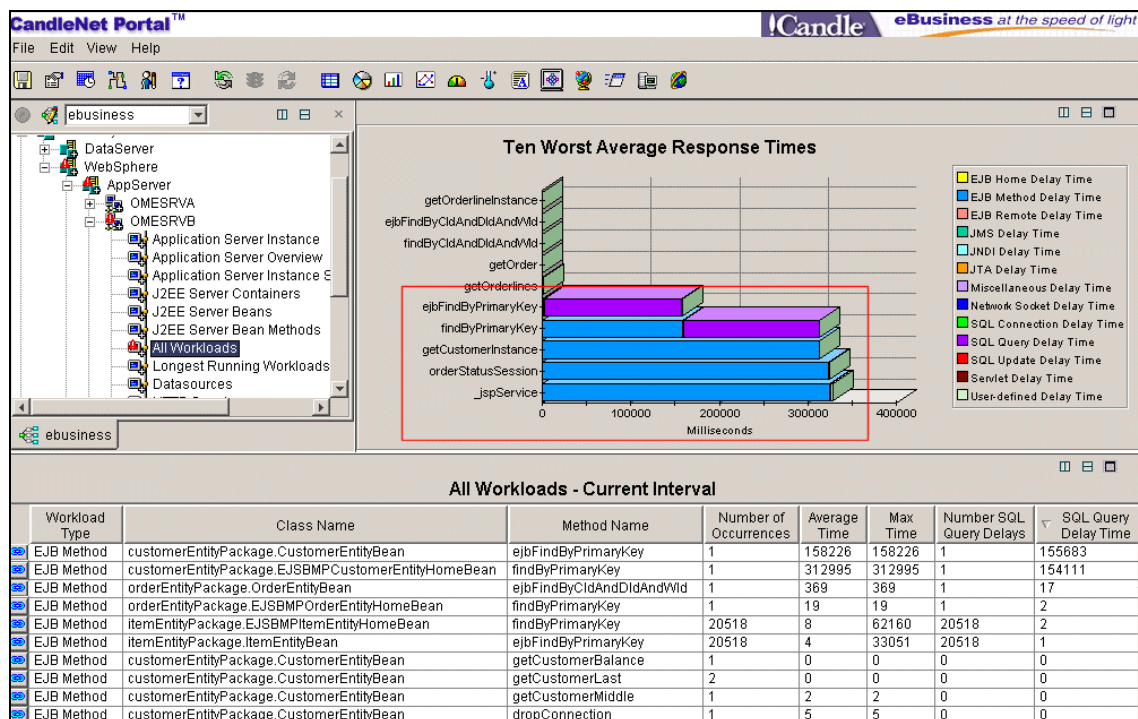


Figure 5-39 All Workloads

The bar chart view in Figure 5-39 shows four workloads with average response times of over 300 seconds (300000 ms). Method

ejbFindByPrimaryKey has an average response of about 150 seconds. All other workloads have sub-second response times.

We use the bar chart legend to analyze what these workloads are doing:

- Three workloads are spending about 300 seconds waiting for other EJB methods (blue on the bar chart).
- Two workloads are spending about 150 seconds waiting for SQL queries (purple on the bar chart).

We assume that the SQL queries are the bottleneck, causing the other workloads to wait.

4. Looking more closely at the table view on the All Workloads workspace from Figure 5-39 on page 196, we sort the table in descending order by SQL Query Delay Time in order to identify poorly performing SQL queries.

Method Name	Number of Occurrences	Average Time	Max Time	Number SQL Query Delays	SQL Query Delay Time
ejbFindByPrimaryKey	1	158226	158226	1	155683
findByPrimaryKey	1	312995	312995	1	154111
ejbFindByCIdAndDIdAndWId	1	369	369	1	17
findByPrimaryKey	1	19	19	1	2
findByPrimaryKey	20518	8	62160	20518	2
ejbFindByPrimaryKey	20518	4	33051	20518	1

Figure 5-40 All Workloads sorted by SQL Query Delay Time

Figure 5-40 shows that method `ejbFindByPrimaryKey` has the highest SQL Query Delay Time (155683 ms). We want to know if the long response time for `ejbFindByPrimaryKey` is due to many short SQL queries, or perhaps one long query.

5. Click the link button on the table row for `ejbFindByPrimaryKey` to display the Selected Workload Delays workspace. Figure 5-41 on page 198 shows detailed delay information for method `ejbFindByPrimaryKey`.

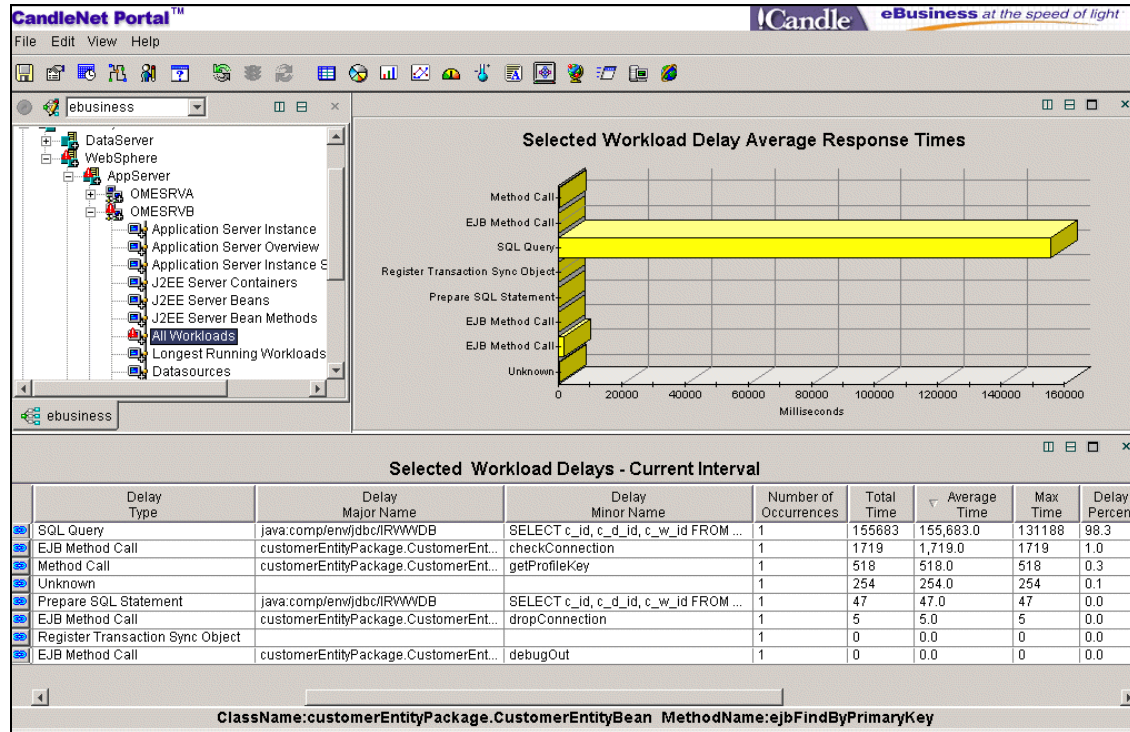


Figure 5-41 Selected Workload Delays

Looking more closely at the table view on the Selected Workload Delays workspace in Figure 5-41, we sort in descending order by Average Time to find the longest delay for method `ejbFindByPrimaryKey`. Figure 5-42 shows the largest delay to method `ejbFindByPrimaryKey` is a SELECT statement to datasource IRWWDB, with an average response time of over 155 seconds (15,683.0 ms).

Delay Major Name	Delay Minor Name	Number of Occurrences	Average Time	CPU Time
java:comp/env/jdbc/IRWWDB	SELECT c_id, c_d_id, c_w_id FROM CBIVP.customer ...	1	155,683.0	7,093.871
customerEntityPackage.CustomerEnt...	checkConnection	1	1,719.0	1,009.311
customerEntityPackage.CustomerEnt...	getProfileKey	1	518.0	134.464

Figure 5-42 Selected Workload Delays table view

It appears that we have a problem in DB2, but we can't tell if DB2 is being impacted by another application in the sysplex. Use OMEGAMON XE for DB2 to investigate the performance of DB2.

- From the ebusiness navigator tree, select the **Detailed Thread Exception** workspace for Inventory Control's datasharing group to see DB2 threads.

Detailed Thread Exceptions													
	Plan Name	Correlation Identifier	Connection Identifier	DB2ID	Elapsed Time	Prefetch Rate	Read I/O Rate	Resource Limit Percent	Wait Time Drain Claims	Wait Time Drain Lock	Wait Time Global Lock	Wait Time Log Queue	Wait Time Procedure
	DSNJDBC	OMESRVS	RRSAF	DB4C	00:05:17	22.1	692.9	0.0	0.000	0.000	0.000	0.000	0.000
	DSNJDBC	OMESRVS	RRSAF	DB4C	01:07:51	0.0	0.0	0.0	0.000	0.000	0.000	0.000	0.000
	DSNJDBC	OMESRVS	RRSAF	DB4C	00:00:00	0.0	0.0	0.0	0.000	0.000	0.000	0.000	0.000
	DSNJDBC	OMESRVS	RRSAF	DB4C	00:00:00	0.0	0.0	0.0	0.000	0.000	0.000	0.000	0.000

Figure 5-43 Detailed Thread Exceptions workspace

Looking more closely at the table view in Figure 5-43, we filter by Correlation ID = OMESRVS to locate the DB2 threads used by our application server.

Plan Name	Correlation Identifier	Connection Identifier	DB2ID	Elapsed Time	Prefetch Rate	Read I/O Rate
DSNJDBC	OMESRVS	RRSAF	DB4C	00:05:17	22.1	692.9
DSNJDBC	OMESRVS	RRSAF	DB4C	01:07:51	0.0	0.0

Figure 5-44 Detailed Thread Exceptions table view

Figure 5-44 shows our application server (OMESRVS), using two DB2 threads for DB2ID=DB4C. We know that DB4C is on the same LPAR as server instance OMESRVB. We see that one of the threads for DB2 plan DSNJDBC has a high Read I/O Rate (692.9). We want to understand if DB2 is reading from the buffer pool or performing physical I/O.

- Select **Volume Activity** from the ebusiness navigator tree to view physical I/O rates by volume. Refer to Figure 5-43 on page 199 to access Volume Activity.

Volume Name	Utilization	Service Time	Total I/O	DB2 I/O	Total I/O Rate	DB2 I/O Rate
TOTDBR	0	2.0	16	0	0.4	0.0
ERW00B	265	0.9	12538	12538	357.7	357.7
ERW017	39	0.9	5089	5089	145.1	145.1
ERW011	0	0.0	0	0	0.0	0.0

Figure 5-45 Volume Activity table view

Figure 5-45 shows an extract from the table view of the Volume Activity workspace. We see that DB2 is causing high physical I/O rates to volumes ERW00B and ERW017. The I/O rates for both volumes have exceeded the product-provided thresholds.

It appears that DB2 is not able to process the high rate of read requests from the buffer pool. Note that Total I/O is the same as DB2 I/O. This means that no other workload is causing contention on these volumes. We deduce that a specific SELECT statement from method `ejbFindByPrimaryKey` is causing DB2 to perform excessive physical I/O reads.

If OMEGAMON XE for DB2plex were installed, there would be a link button for each row on the Volume Activity table view. The link navigates to a workspace that shows which databases are using that volume. We could then drill down from the databases to identify which tablespaces are using that volume. This would show us which DB2 tables are causing our problem.

At this point we need to turn the problem over to the DB2 administrator to diagnose the problem. The DB2 administrator could use OMEGAMON II for DB2 to start an application trace to determine whether a tablespace scan was occurring. We suspect that this database needs an index to eliminate the tablespace scan.

5.4.5 Example 7 - Transaction hang or time-out

Users complain that one of their business functions is timing out. In this example:

- ▶ We receive an alert on the navigator tree that average response times for some workloads have exceeded the predetermined threshold.
- ▶ Using the All Workloads workspace we see that one method is experiencing delays for JMS requests.
- ▶ We drill down to the Selected Workload Delays workspace and see that this method is consistently taking 180 seconds for JMS Message Get requests. We suspect a JMS timeout.
- ▶ We select the **JMS Summary** workspace to determine whether all JMS requests are experiencing delays or just the Message Get requests for this

workload. We see that Put requests to the same queue manager are being processed normally.

- ▶ We deduce that the method is putting a message on one queue and timing out waiting for the response on the reply-to queue.
- ▶ We use OMEGAMON XE for WebSphere MQ to investigate the problem with the MQ queues. We select the **Queue Statistics** workspace and see that the trigger queue is not being processed because it is Get disabled.

Procedure

1. We receive an alert on the ebusiness navigator tree. Position the mouse pointer over the alert (red triangle icon) to show details. Figure 5-46 shows that the EIS application has a critical alert for server instance OMTSRVA. The alert `WAS_Workload_AvgResp_Critical` indicates that one or more workloads have exceeded the threshold for average response time.

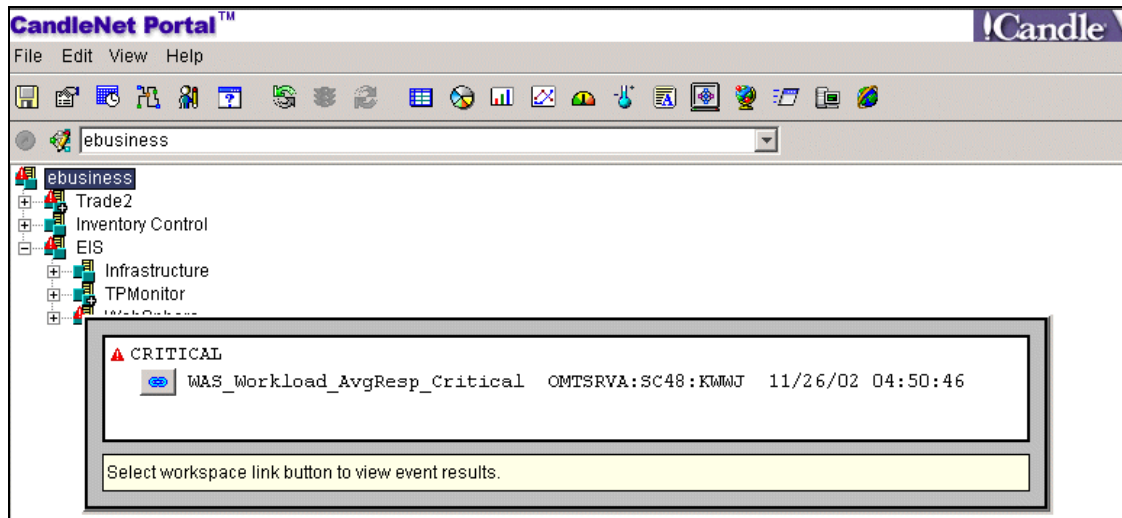


Figure 5-46 Alert on the ebusiness navigator tree

2. Click the link button for `WAS_Workload_AvgResp_Critical` in the critical alert window to display the Current Situation Values table.

Average Time	Server Name	Workload Type	Class Name	Method Name	Number of Occurrences	JMS Delay Time	JMS Delay Percent
181663	OMTSRVA	Servlet	_JMSResults_jsp_3	_jspService	18	181152	99.7

Figure 5-47 Current situation values

Figure 5-47 on page 201 displays one servlet that has exceeded the response time threshold. This servlet has an average response of over 181 seconds (181663 ms), with the majority of its time spent waiting for JMS (99.7%).

Clicking the link button in Figure 5-37 on page 195 also expands the navigator tree to display the All Workloads selection for instance OMTSRVA.

3. Select the All Workloads workspace for instance OMTSRVA.

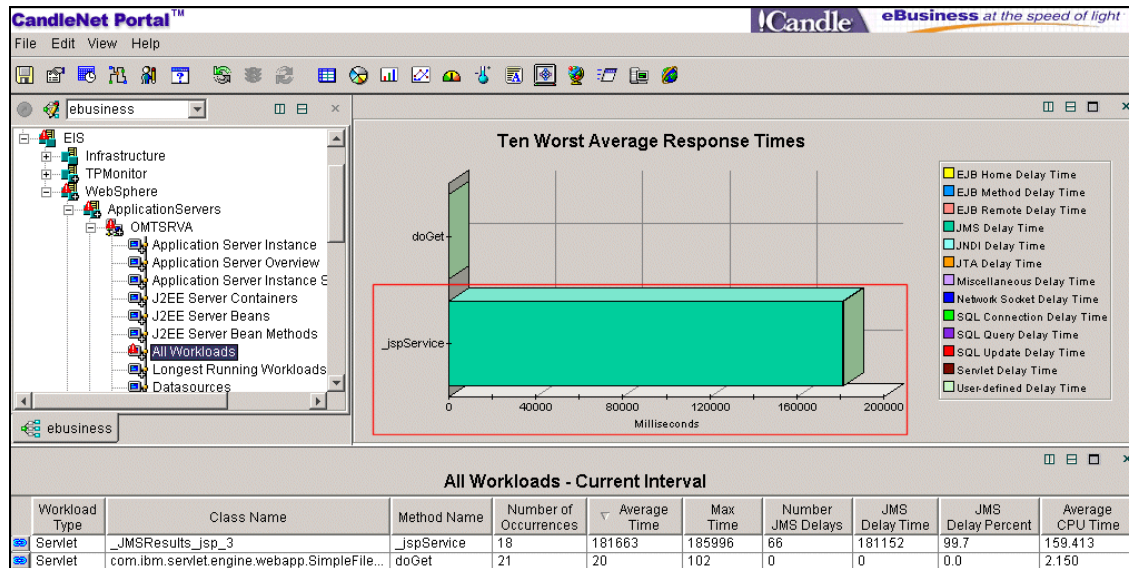


Figure 5-48 All Workloads workspace

The bar chart view from Figure 5-48 confirms that one workload has an average response time of over 180 seconds. All other workloads have sub-second response times. We use the bar chart legend to confirm that this workload is waiting for JMS requests (green on the bar chart).

4. Looking more closely at the table view on the All Workloads workspace from Figure 5-48, we sort the table in descending order by Average time.

Method Name	Number of Occurrences	Average Time	Max Time	Number JMS Delays	JMS Delay Time	JMS Delay Percent
_jspService	18	181663	185996	66	181152	99.7
doGet	21	20	102	0	0	0.0

Figure 5-49 All Workloads table view

Figure 5-49 shows that there were 18 occurrences of method _jspService in the collection interval with an average response time of 181663 ms. We note

that the maximum response time was 185996 ms. This means that every occurrence of this workload had a response time close to 180 seconds.

We drill down to understand which JMS requests are causing this 180 second delay.

- Click the link button on the All Workloads table row for method `_jspService` to display the Selected Workload Delays workspace:

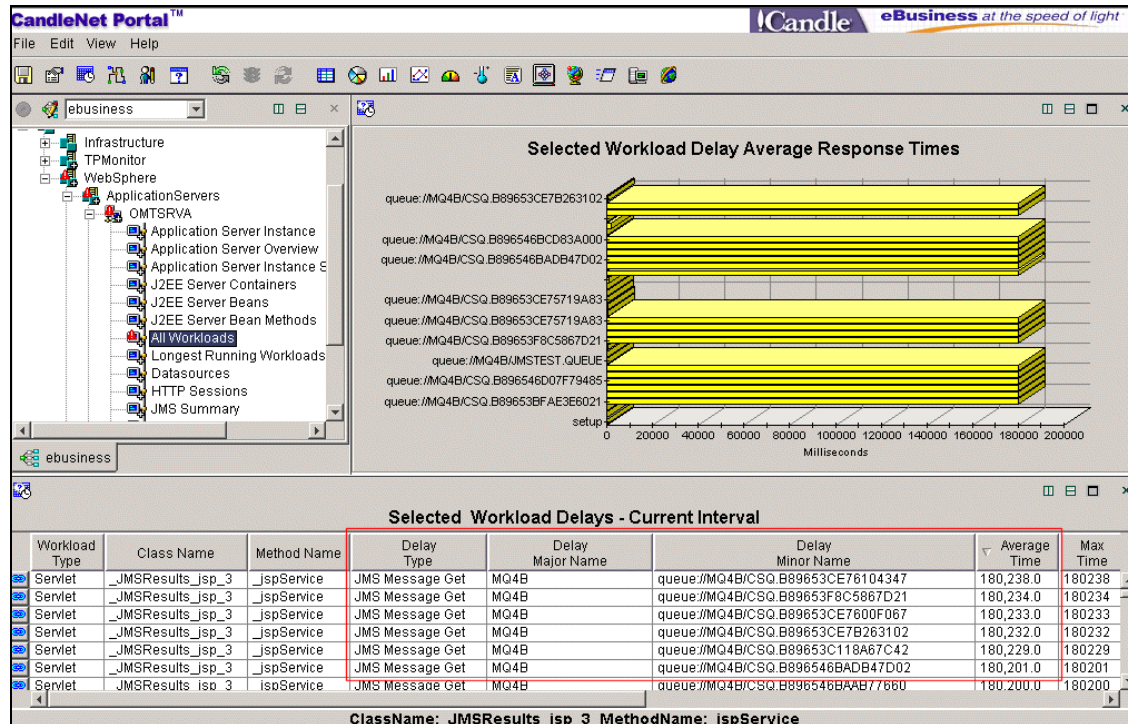


Figure 5-50 Selected Workload Delays

The table view in Figure 5-50 shows that method `_jspService` is performing JMS Message Get requests to queue manager MQ4B. The response time for each JMS Message Get request is almost identical (for example 180238ms, 180234ms, and 180233ms).

It appears that this workload may be experiencing time-outs on JMS Message Get requests after 180 seconds. We want to understand whether *all* JMS requests are experiencing delays or just JMS Message Get requests for this particular workload.

- From the ebusiness navigator tree select the **JMS Summary** workspace.

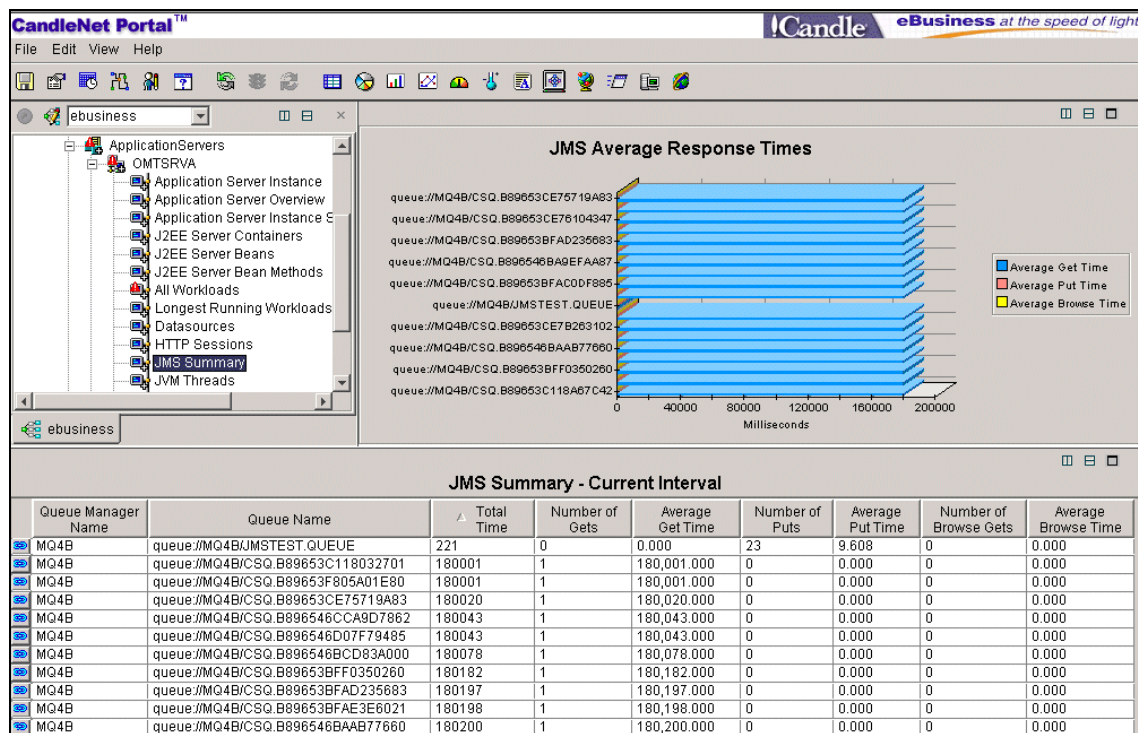


Figure 5-51 JMS Summary

Looking more closely at the table view in Figure 5-52 shows that there are 23 Put requests for queue name JMSTEST.QUEUE with an average response of 9.608 ms. These sub-second Put requests are being issued to the same queue manager (MQ4B) as the 180-second Get requests. It appears that only Get requests are experiencing delays.

Queue Manager Name	Queue Name	Total Time	Number of Gets	Average Get Time	Number of Puts	Average Put Time
MQ4B	queue://MQ4B/JMSTEST.QUEUE	221	0	0.000	23	9.608
MQ4B	queue://MQ4B/CSQ.B89653C118032701	180001	1	180,001.000	0	0.000
MQ4B	queue://MQ4B/CSQ.B89653F805A01E80	180001	1	180,001.000	0	0.000

Figure 5-52 JMS Summary table view

We deduce that this workload is putting a message on queue JMSTEST.QUEUE, then creating a reply-to queue and issuing a Get for the response. For some reason the process that retrieves the message from JMSTEST.QUEUE and returns the response on the reply queue is failing. We use OMEGAMON XE for WebSphere MQ to investigate queue manager MQ4B.

- From the ebusiness navigator tree select the **Queue Statistics** workspace for queue manager MQ4B.

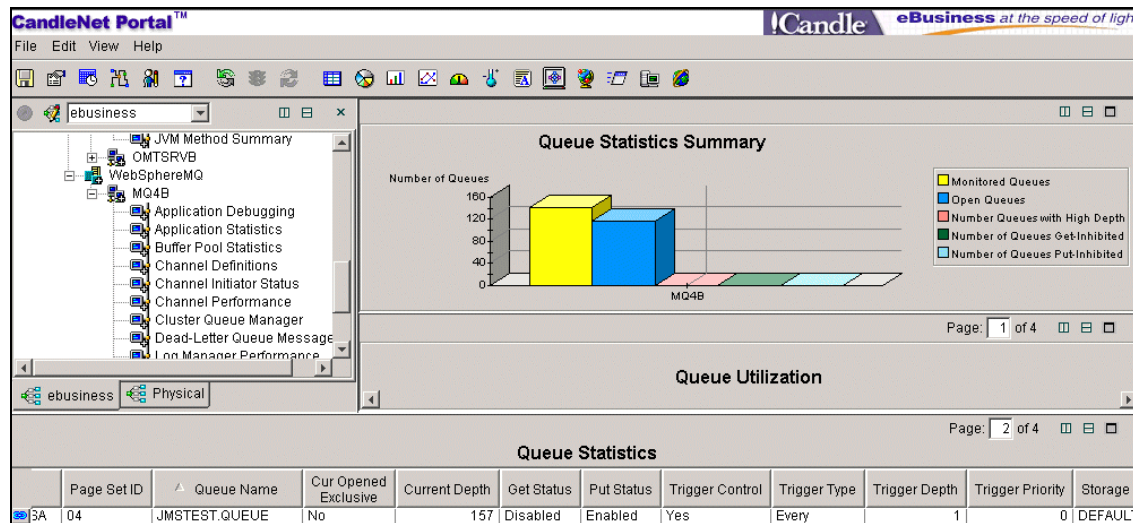


Figure 5-53 Queue Statistics

Looking more closely at the table view from Figure 5-53, we filter for queue JMSTEST.QUEUE. Figure 5-54 shows that queue JMSTEST.QUEUE has a trigger depth of 1, but a queue depth of 157. This is caused by the Get Status of Disabled.

Queue Name	Cur Opened Exclusive	Current Depth	Get Status	Put Status	Trigger Control	Trigger Type	Trigger Depth
JMSTEST.QUEUE	No	157	Disabled	Enabled	Yes	Every	1

Figure 5-54 Queue Statistics table view

In this example, the process that is supposed to retrieve the message from queue JMSTEST.QUEUE is unable to get the message. This means that the response is never put on the reply queue, resulting in a timeout for the Get requests for the WebSphere application.

Example 7 - Advanced procedure

An alternative to the previous procedure is to use the custom business view for the PRR application. With this approach we can proactively monitor all the key components of the application from one workspace.

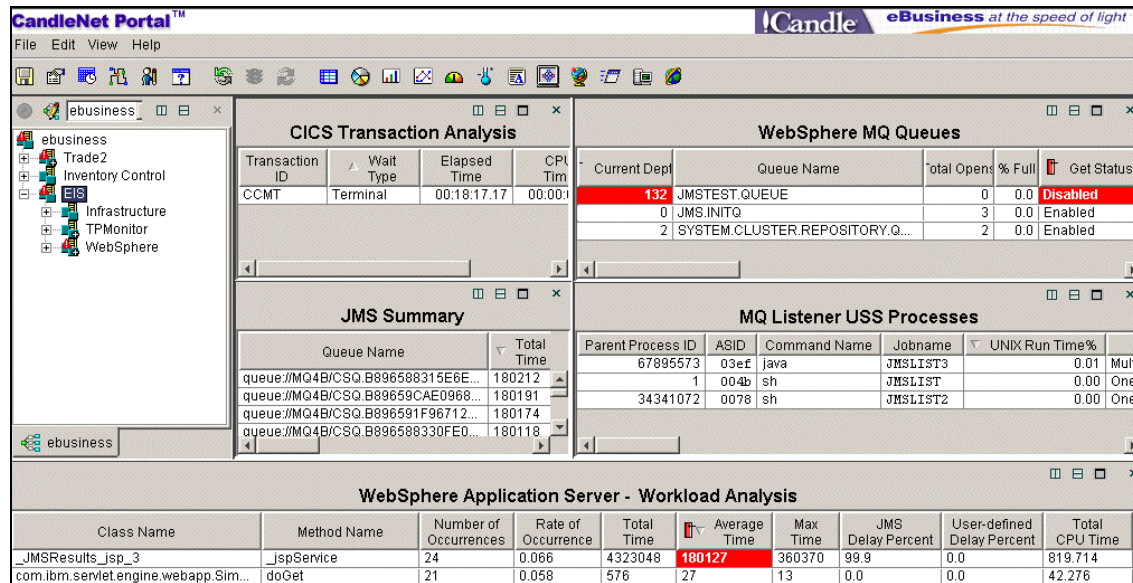


Figure 5-55 Custom business view

Figure 5-55 shows the custom business view that we configured for application PRR. It uses OMEGAMON DE to integrate the following information into a single workspace:

- The WebSphere MQ Queues view displays information from queue manager MQ4B on queue JMSTEST.QUEUE used by the PRR application. Based on our previous analysis we know that the trigger depth is one message, so we configured this view to alert us when the queue depth is greater than 3. We also configured this view to alert us when the queue is Get disabled.
- The JMS Summary view displays response time information for JMS requests for server instance OMTSRVA.
- The MQ Listener USS Processes view displays information on the USS processes that get the messages from the MQSeries trigger queue and return data on the reply-to queues for the PRR application.
- The WebSphere Application Server - Workload Analysis view displays average response times for workloads on server instance OMTSRVA. We configured this view to alert us when the average response time for a workload is greater than 3 seconds.

The custom view for application PRR also includes a CICS Transaction Analysis view that is not pertinent to this example. This view is used by another business function within the PRR application; refer to “Example 4 - Advanced procedure” on page 193 for more details.

Looking more closely at the WebSphere MQ Queues view in Figure 5-56, we see two alerts for the MQ trigger queue (JMSTEST.QUEUE) used by application PRR. The queue is Get disabled, resulting in a backlog of messages on the queue.

Current Dept	Queue Name	Total Opens	% Full	Get Status
132	JMSTEST.QUEUE	0	0.0	Disabled
0	JMS.INITQ	3	0.0	Enabled

Figure 5-56 WebSphere MQ Queues view

The WebSphere Application Server - Workload Analysis view shown in Figure 5-57 shows that application PRR has a workload that is experiencing average response times of over 180 seconds (180127 ms).

Class Name	Method Name	Number of Occurrences	Rate of Occurrence	Total Time	Average Time
_JMSResults_jsp_3	_jspService	24	0.066	4323048	180127

Figure 5-57 WebSphere Application Server - Workload Analysis view

In one workspace, the custom business view for application PRR shows long response times for server instance OMTSRVA and issues with the MQ trigger queue used by this application.

5.4.6 Example 8 - Static pages serving

The purpose of this example is to detect when our WebSphere Application Server on z/OS is serving static pages.

This example is not intended to imply that WebSphere for z/OS is not a viable choice for serving static pages, but in our environment we expect the WebSphere Edge Server to be handling the static content. Our objective is to detect a configuration error or failover condition for our edge server.

WebSphere handles static pages using an IBM-supplied class: `com.ibm.servlet.engine.webapp.SimpleFileServlet`, so we define a warning situation that detects when this class is executed.

In this example:

- We receive warning alerts on the ebusiness navigator tree from all three of our server instances, indicating that they are serving static pages.
- Selecting the **All Workloads** workspace for one of our server instances, we confirm that `SimpleFileServlet` is being executed many times.

- We want to understand the impact to WebSphere of serving these static pages, so we review the amount of CPU used by SimpleFileServlet.
- We use OMEGAMON XE for OS/390 UNIX System Services to see if there is any increase in HFS I/O as a result of handling the static content.
- We use OMEGAMON XE for Mainframe Networks to review the load on TCP/IP.

Procedure

1. We receive a warning alert on the ebusiness navigator tree. Position the mouse pointer over the alert (yellow triangle icon) to show details.

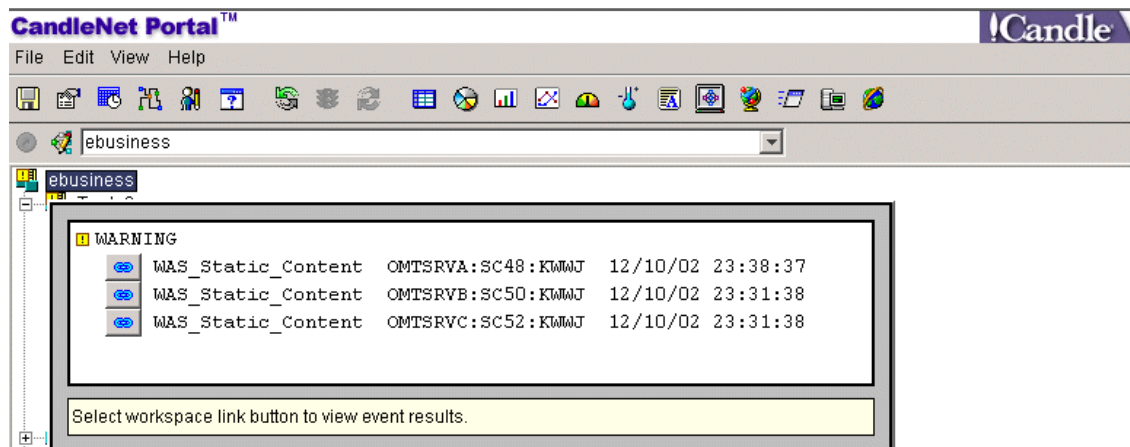


Figure 5-58 Warning alerts on the ebusiness navigator

Figure 5-58 shows that the Trade2 application has three warning alerts indicating that static pages are being handled by server instances OMTSRVA, OMTSRVB, and OMTSRVC.

Since the alert is the same for all three instances, it doesn't matter which alert we select. We will analyze instance OMTSRVA.

2. Click the first link button for WAS_Static_Content in the warning alert window to display the Current Situation Values table for instance OMTSRVA.

Class Name	Server Name	Workload Type	Method Name	Number of Occurrences	Average Time
com.ibm.servlet.engine.webapp.SimpleFileServlet	OMTSRVA	Servlet	doGet	1423	27

Figure 5-59 Current situation values

Figure 5-59 confirms that class name `com.ibm.servlet.engine.webapp.SimpleFileServlet` has been invoked by server instance OMTSRVA.

Clicking the link button in Figure 5-58 on page 208 also expands the navigator tree to display the All Workloads selection for instance OMTSRVA.

3. Select the All Workloads workspace for instance OMTSRVA.

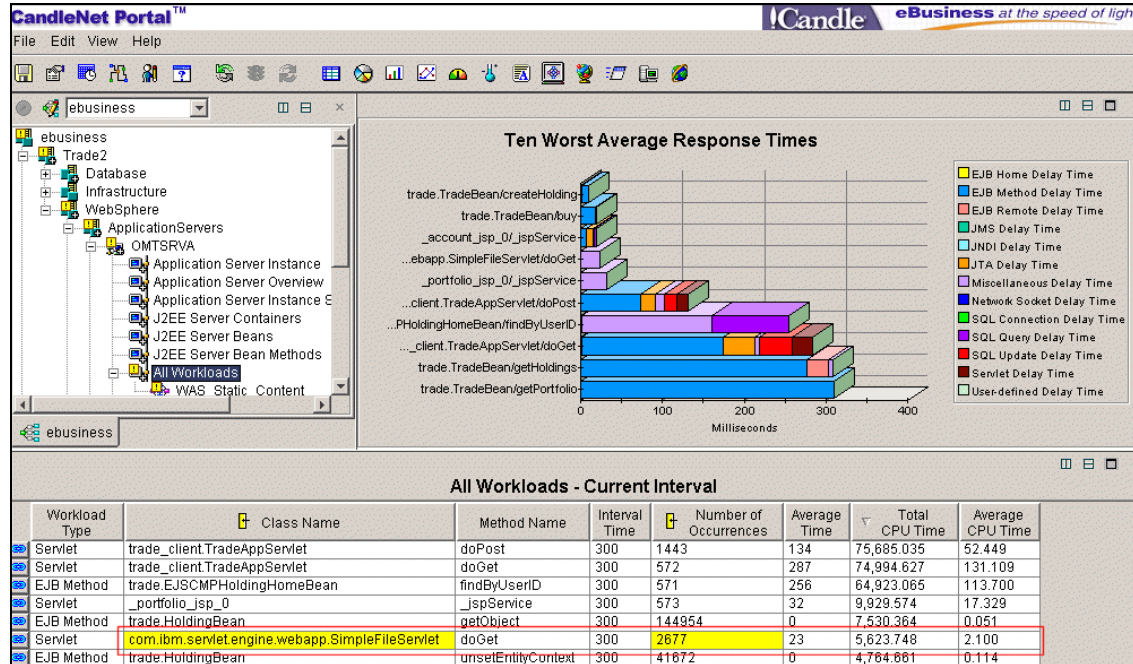


Figure 5-60 All Workloads workspace

We sort the table in descending order by Total CPU Time and locate the entry for SimpleFileServlet.

Class Name	Method Name	Interval Time	Number of Occurrences	Average Time	Total CPU Time	Average CPU Time
trade_client.TradeAppServlet	doPost	300	1443	134	75,685.035	52.449
trade_client.TradeAppServlet	doGet	300	572	287	74,994.627	131.109
trade.EJSCMPHoldingHomeBean	findByUserID	300	571	256	64,923.065	113.700
_portfolio_jsp_0	_jspService	300	573	32	9,929.574	17.329
trade.HoldingBean	getObject	300	144954	0	7,530.364	0.051
com.ibm.servlet.engine.webapp.SimpleFileServlet	doGet	300	2677	23	5,623.748	2.100
trade.HoldingBean	unsetEntityContext	300	41672	0	4,764.661	0.114

Figure 5-61 All Workloads table view

Figure 5-61 on page 209 shows that class `com.ibm.servlet.engine.webapp.SimpleFileServlet` has been invoked 2677 times in the current collection interval, with an Average CPU time of 2.100 ms. The collection Interval Time is 300 seconds, and the Total CPU Time to handle the static pages during this 5-minute period is 5.6 seconds (5,623.748 ms).

The CPU cost does not seem excessive.

4. We use OMEGAMON XE for OS/390 UNIX System Services to determine the impact on HFS I/O rates. Select the Mounted File Systems workspace from the navigator tree view.

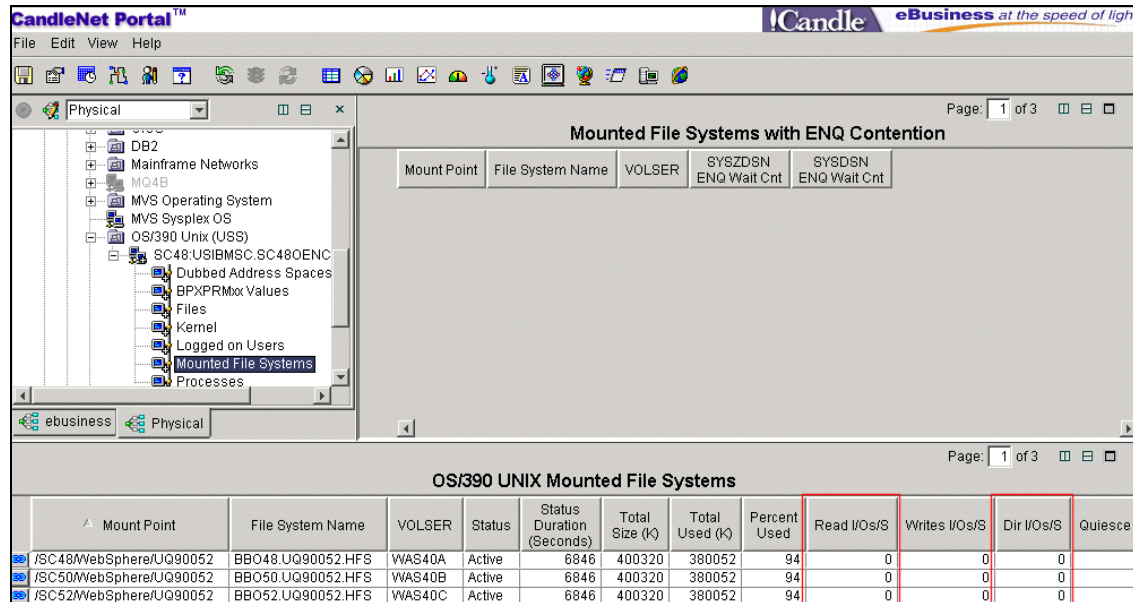


Figure 5-62 OS/390 UNIX Mounted File Systems

We filter the table view in Figure 5-62 on the Mount Points File System workspace associated with our WebSphere instances. We see mount points for all three of our server instances because we have shared HFS. All three server instances have zero Read I/Os per second and zero Directory I/Os per second. We deduce that the static pages are being cached by USS since there is no increase in I/O activity for the WebSphere.

5. We use OMEGAMON XE for Mainframe Networks to determine the impact to TCP/IP. Select the **Network Applications** workspace from the ebusiness navigator tree view.

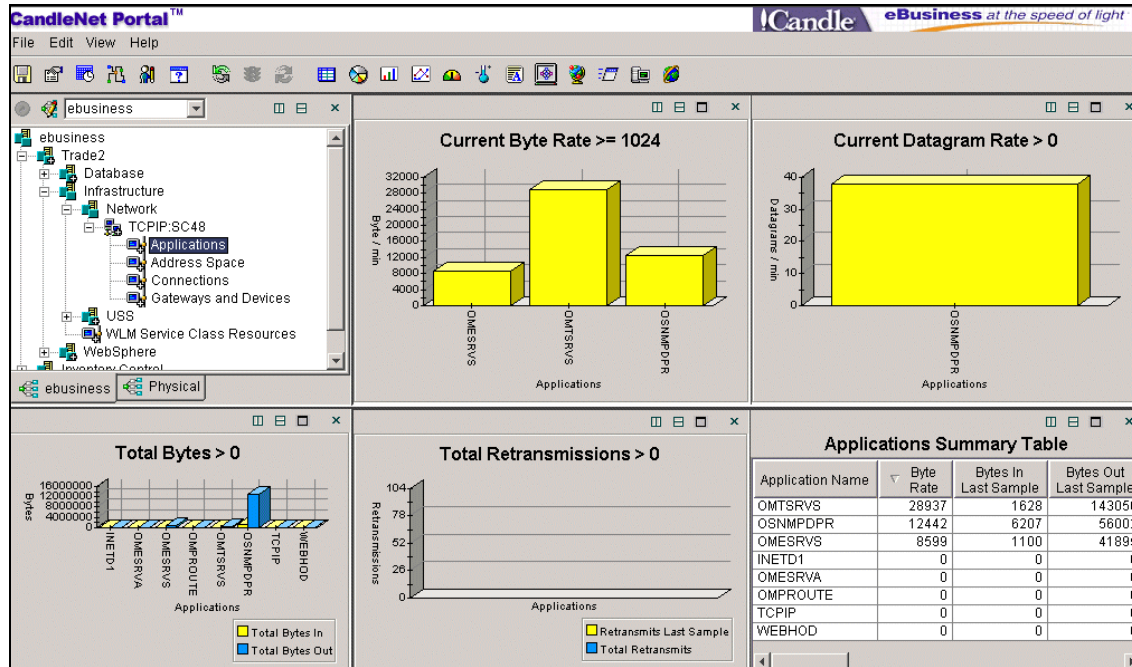


Figure 5-63 Network Applications workspace

Looking more closely at the Applications Summary Table view on the Network Applications workspace in Figure 5-63, we sort in descending order by Byte Rate.

Application Name	Byte Rate	Bytes In Last Sample	Bytes Out Last Sample	Total Bytes Last Sample	Total Bytes In	Total Bytes Out	Total Bytes
OMTSRVS	28937	1628	143056	144684	8030	554519	562549
OSNMPDPR	12442	6207	56002	62209	1440877	13056349	14497226
OMESRVS	8599	1100	41895	42995	18502	679991	698493

Figure 5-64 Network Applications Summary Table view

Figure 5-64 shows that the OMTSRVS server instance has a Byte Rate of 28K per minute (28937).

In our example, the application has a few static pages and they do not contain large graphics. As a result, having our server instances handle the static pages was not a significant impact to CPU, file I/O, or network traffic. In your environment, requests for many different pages with a lot of static content may result in significant overhead. Investigating the impact to these system

resources can help you determine if this is a good choice for your environment.

5.4.7 Example 9 - Increased WebSphere activity

A system operator notices that system resource usage for a particular WebSphere Application Server region has increased over the last hour.

In this example:

- ▶ We do not see any alerts for WebSphere or its connected components on the navigator tree.
- ▶ Using the All Workloads workspace, we see that workloads are being processed with sub-second response times. The application server instance appears to be performing normally.
- ▶ We select a workload and drill down to the Selected Workload - History workspace to see how its response time has varied over the last hour. We see that the number of occurrences of this workload has increased significantly in the last hour, and the average response times have been increasing linearly.
- ▶ Using the HTTP Sessions workspace, we see that the overall throughput for this server instance has grown significantly over the last hour. We deduce that increased demand is the cause of the increased system resource usage.
- ▶ Using the Application Server Overview workspace, we review the JVM memory use over the last hour. We confirm that there is no memory leak and that there is sufficient memory for further increase in throughput.

Procedure

1. We are notified by a system operator that a server region for the Trade2 application has increased its use of system resources over the last hour. It is possible that we have a memory leak and the increase in resource usage is due to increased garbage collections.

There are no alerts for WebSphere or any of the connected components on the ebusiness navigator tree. Since there are no response time alerts for this instance, workloads are either performing within response time goals, or possibly the region is stalled and no work is being processed. Select **All Workloads** on the ebusiness navigator tree for the Trade2 application.

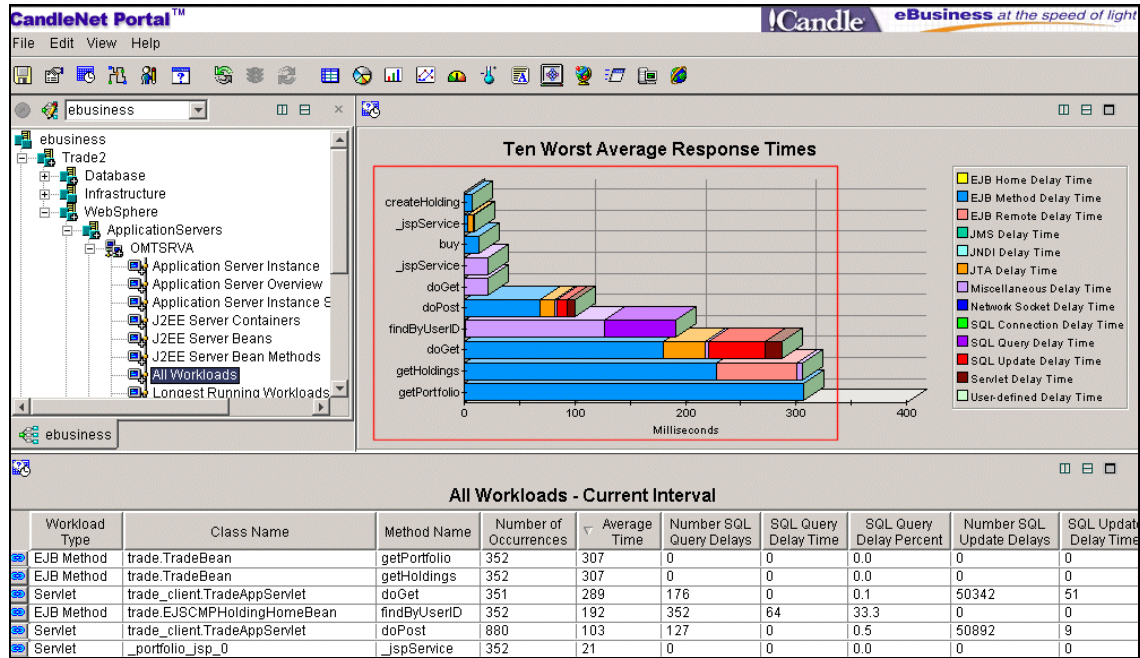


Figure 5-65 All Workloads workspace

The bar chart view in Figure 5-65 shows that all workloads have average response times of less than 300 ms. Looking more closely at the table view for the All Workloads workspace, we sort in descending order by Average Time:

Workload Type	Class Name	Method Name	Number of Occurrences	Average Time	Number SQL Query Delays	SQL Query Delay Time
EJB Method	trade.TradeBean	getPortfolio	352	307	0	0
EJB Method	trade.TradeBean	getHoldings	352	307	0	0
Servlet	trade_client.TradeAppServlet	doGet	351	289	176	0
EJB Method	trade.EJSCMPHoldingHomeBean	findByUserID	352	192	352	64

Figure 5-66 All Workloads table view

Figure 5-66 shows that the server instance has processed more than 352 occurrences of the worst performing workloads in the current collection interval (5 minutes). We also see that EJB Method `findByUserID` has an average response of 64 ms for SQL Query requests, so we do not appear to have a database problem.

We have confirmed that this server instance is not stalled and is processing workloads in a timely manner.

We have been told that this server instance has increased its use of system resources in the last hour, so we want to determine if the average response times for the workloads have also been increasing.

It does not matter which workload we choose, but EJB method `findByUserID` is a good candidate because it contains SQL queries, so its response time could be affected by contention caused by increased load.

2. Either click the link button on the All Workloads table row for method `findByUserID`, or right-click the bar representing `findByUserID` in the All Workloads bar chart view to navigate to the Selected Workload - History workspace. The Selected Workload Average Response Times view in Figure 5-67 shows that the average response time for EJB method `findByUserID` has increased from about 140 ms to about 190 ms over the last hour.

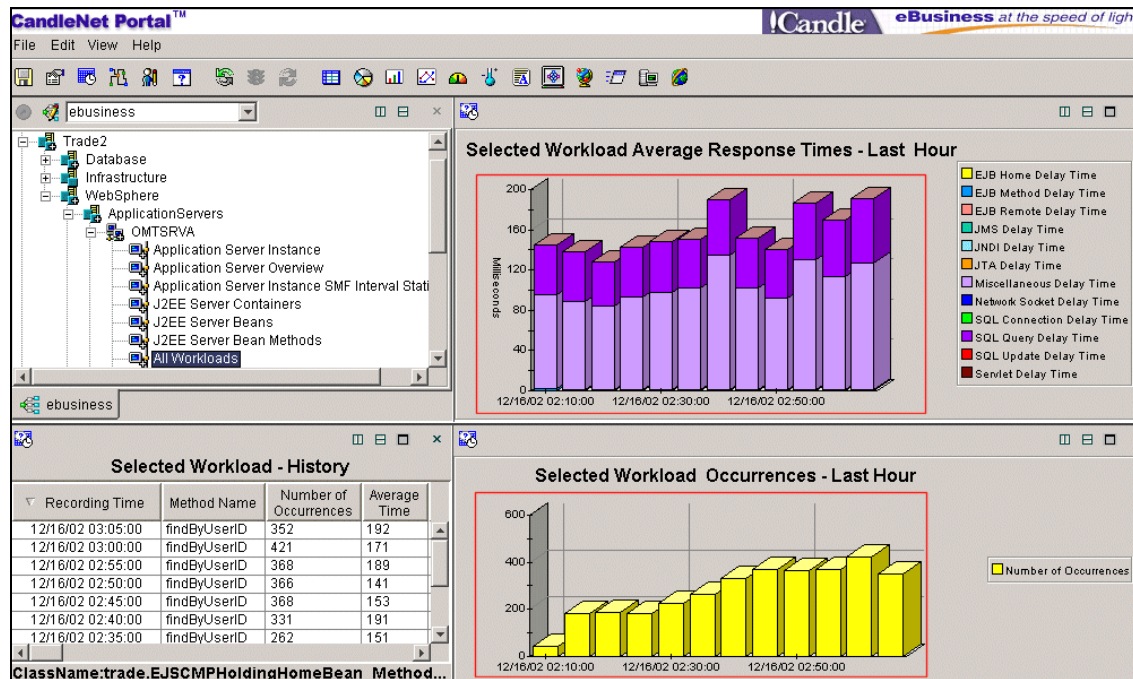


Figure 5-67 Selected Workload - History workspace

The Selected Workload Occurrences view shows that the number of occurrences (in each 5-minute interval) for EJB method `findByUserID` has increased from less than 50 to about 350 in the last hour.

As the load steadily increased, the average response also increased fairly linearly. In fact, the increase in average response time is relatively small (<50% increase) compared to the increase in throughput (> 600%).

We want to understand whether the overall throughput for this server instance has also increased significantly in the last hour.

3. Select **HTTP Sessions** on the navigator tree view. Figure 5-68 shows that the number of HTTP Sessions over the last hour has also grown significantly, from about 20 to 180 sessions.

It appears that the increase in system resources usage over the last hour is due to increased demand for the Trade 2 application.

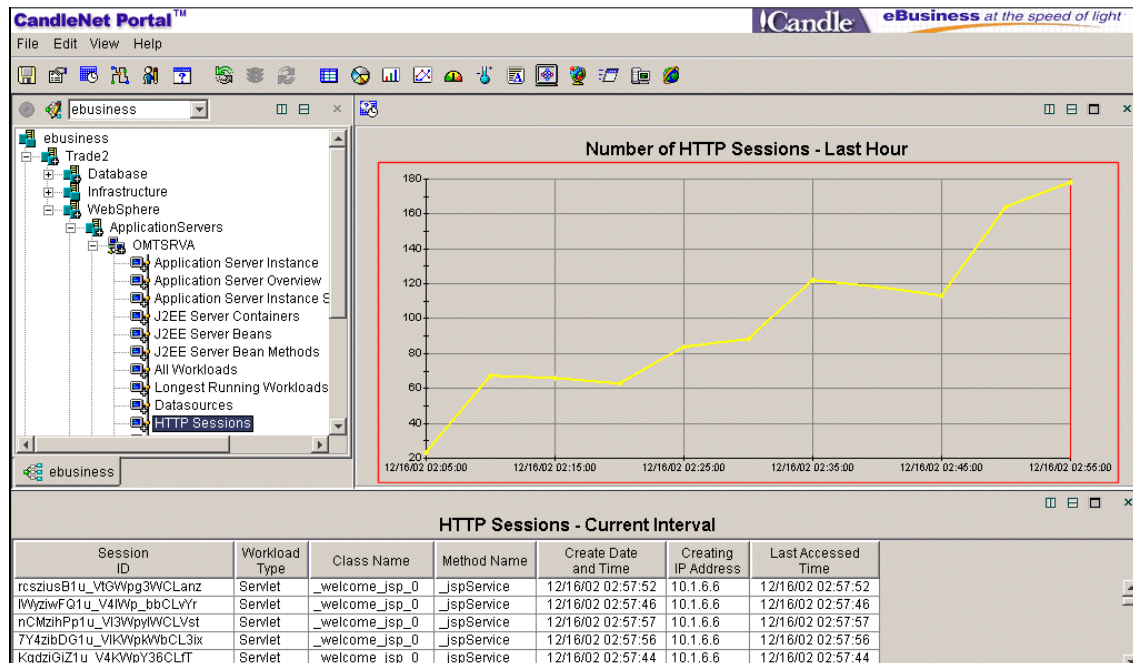


Figure 5-68 HTTP Sessions workspace

Figure 5-67 on page 214 shows that a steady increase in load has resulted in a linear increase in response time. If the load continues to grow, the current server regions will eventually reach a point where they cannot handle the extra load. This could result in significantly longer response times.

JVM memory is one resource that can become exhausted, resulting in poor response time. We want to understand memory use over the last hour to try to determine if we can handle additional load.

4. Select **Application Server Overview** on the navigator tree view. The Application Server Overview table view in Figure 5-69 shows JVM memory use for two server regions for this server instance. We modify the chart view to plot memory use over the last hour for one of the server regions (it doesn't matter which).

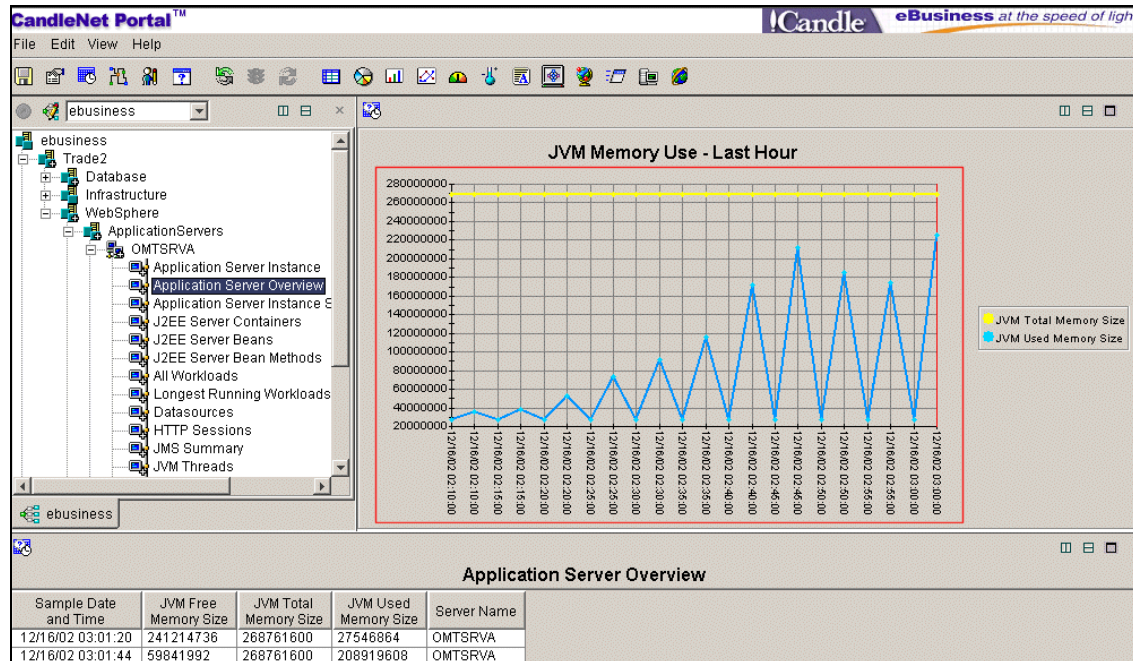


Figure 5-69 Application Server Overview workspace

The chart view in Figure 5-69 shows that maximum JVM memory use has been increasing linearly over the last hour, as throughput has increased. The minimum data points on the graph indicate the memory use after garbage collection. The good news is that the garbage collector has been able to free most of the memory after each collection, and the minimum data points are not increasing over time, which confirms that we do not have a memory leak.

Extrapolating the maximum data points on the graph, it appears that we have sufficient memory for additional throughput. However, we need to continue monitoring memory use, because garbage collections will take longer to run and may run more frequently as our memory use approaches the total memory available.

5.4.8 Example 10 - Identify a method called with high frequency

Users complain that some of the business functions are experiencing poor response times.

In this example:

- ▶ We receive an alert on the navigator tree that average response times for some workloads have exceeded the predetermined thresholds.
- ▶ Using the All Workloads workspace, we see that the workload with the longest average response time is spending most of its time calling user methods and other EJB methods.
- ▶ We drill down to the Selected Workload Delays workspace and see that this method is waiting about 50% of the time for user method debugOut. We suspect a configuration error.
- ▶ We select the Ten Most Frequently Used Workloads workspace and confirm that the debugOut methods from several classes are heavily used.

Procedure

1. We receive an alert on the ebusiness navigator tree. Position the mouse pointer over the alert (red triangle icon) to show details. Figure 5-70 shows that the Inventory Control application has a critical alert for server instance OMESRVA. The alert WAS_Workload_AvgResp_Critical indicates that one or more workloads have exceeded the threshold for average response time.

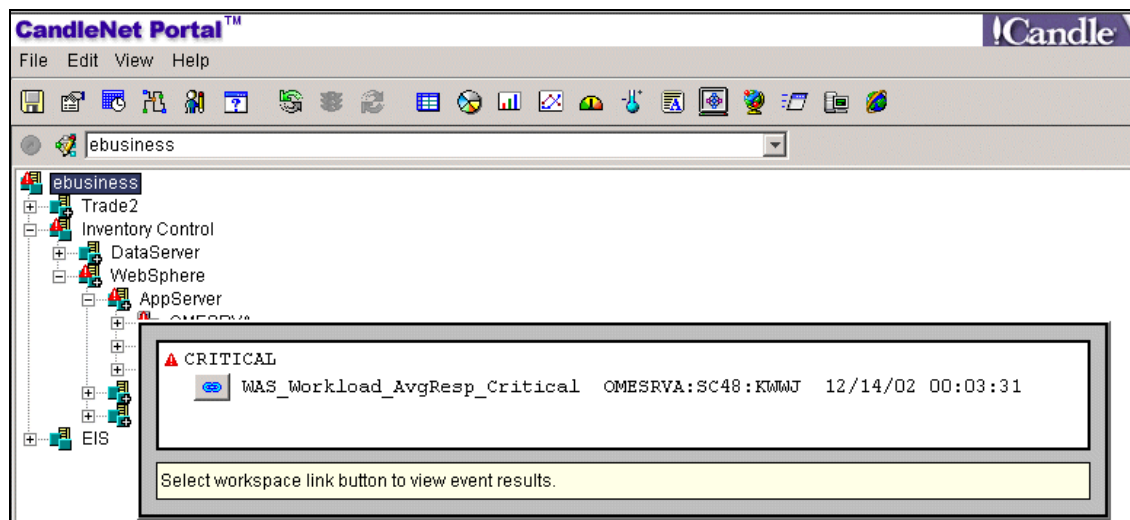


Figure 5-70 Alert on the ebusiness navigator tree

- Click the link button for WAS_Workload_AvgResp_Critical in the critical alert window to display the Current Situation Values table. Figure 5-71 displays two EJB methods and two servlets that have exceeded the response time threshold. For example, the deliverySession method has an average response of almost 7 seconds (6937 ms).

Average Time	Server Name	Workload Type	Class Name	Method Name	Number of Occurrences
6937	OMESRVA	EJB Method	deliverySessionPackage.DeliverySessionBe...	deliverySession	5
6867	OMESRVA	Servlet	_DEAGResults_jsp_3	_jspService	6
3485	OMESRVA	Servlet	_NOAGResults_jsp_3	_jspService	79
3370	OMESRVA	EJB Method	neworderSessionPackage.NewOrderSessio...	newOrderSession	79

Figure 5-71 Current situation values

- Click the link button in Figure 5-70 on page 217 to expand the navigator tree to display the All Workloads selection for instance OMESRVA. Select the All Workloads workspace for instance OMESRVA.

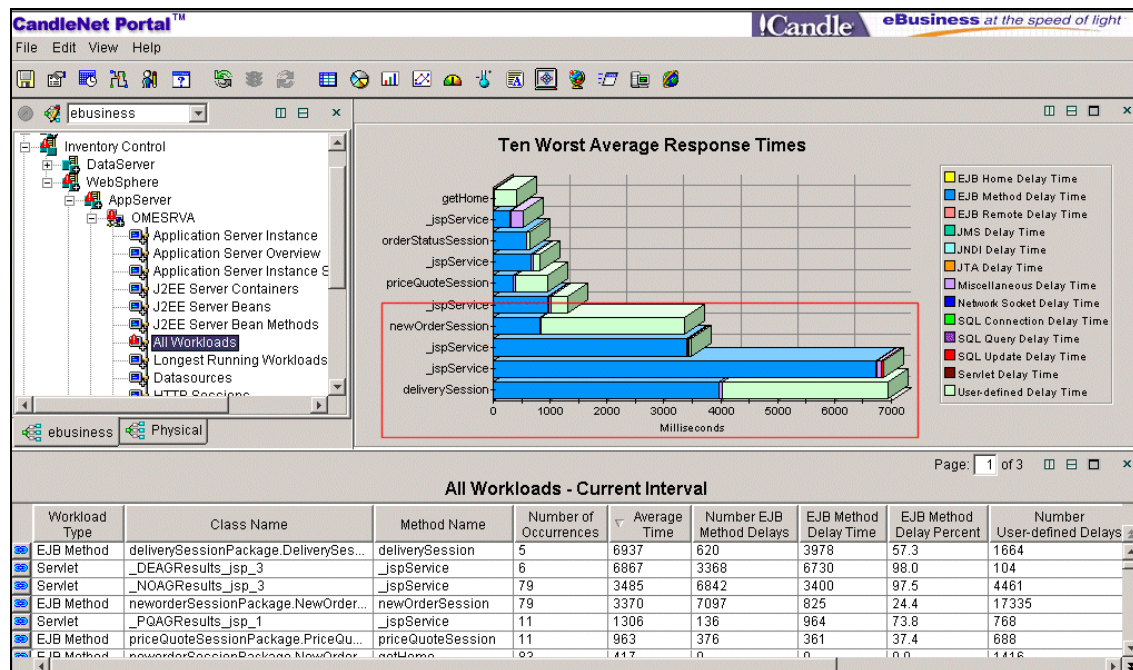


Figure 5-72 All Workloads workspace

The bar chart view on the All Workloads workspace in Figure 5-72 confirms that four workloads have average response times of over 3 seconds.

We use the bar chart legend to see that these workloads are primarily waiting for EJB methods and user-defined delays (blue and aqua on the bar chart).

We drill down on the workload with the longest response time to understand what is causing this delay.

4. Either click the link button on the All Workloads table row for method `deliverySession`, or right-click the bar representing `deliverySession` in the All Workloads bar chart view to display the Selected Workload Delays workspace:

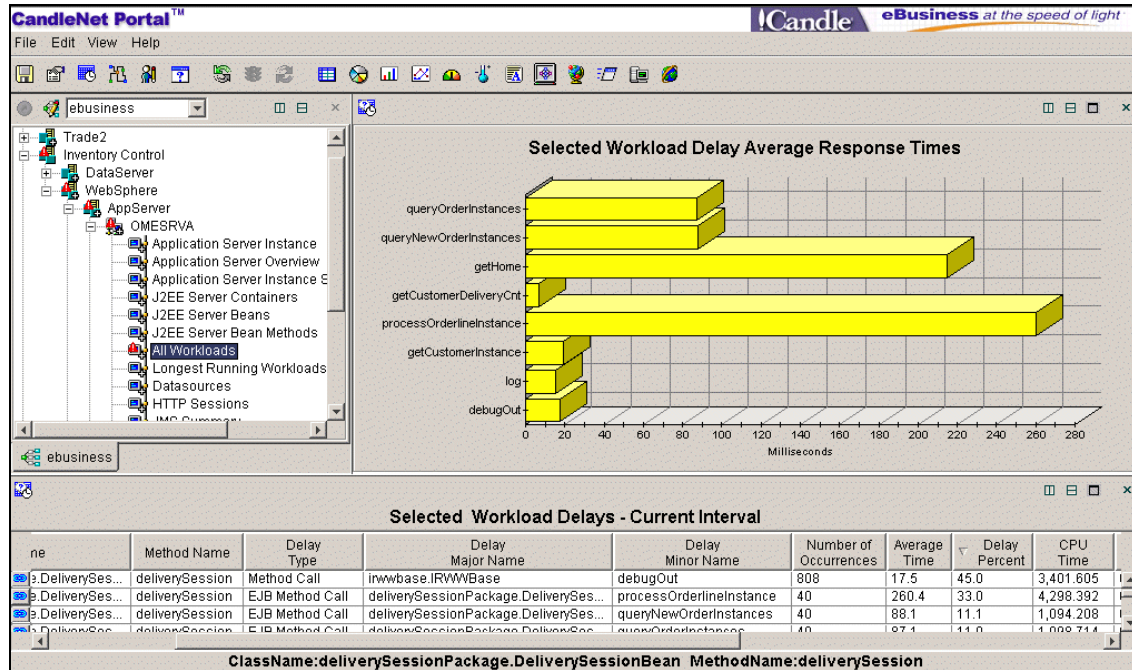


Figure 5-73 Selected Workload Delays

Figure 5-73 shows detailed delay information for method `deliverySession`. Looking more closely at the table view on the Selected Workload Delays workspace, we sort in descending order by Delay Percent.

Delay Type	Delay Major Name	Delay Minor Name	Number of Occurrences	Average Time	Delay Percent
Method Call	irwwwbase.IRWWWBase	debugOut	808	17.5	45.0
EJB Method Call	deliverySessionPackage.DeliverySes...	processOrderlineInstance	40	260.4	33.0
EJB Method Call	deliverySessionPackage.DeliverySes...	queryNewOrderInstances	40	88.1	11.1

Figure 5-74 Selected Workload Delays table view

Figure 5-74 on page 219 shows that method `deliverySession` makes an average of 808 calls to method `debugOut`, accounting for 45.0% of its response time. Based on the method name `debugOut`, we suspect that this workload is performing unnecessary logging.

- From the `ebusiness` navigator tree, right-click `All Workloads` and select the `Ten Most Frequently Used Workloads` workspace.

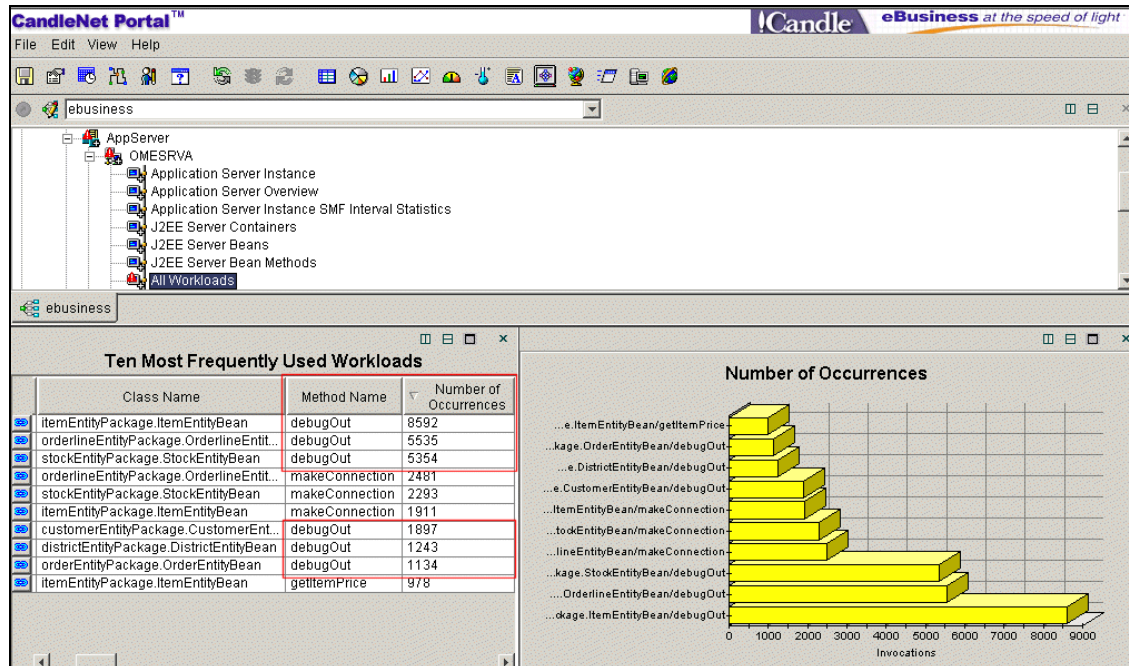


Figure 5-75 Ten Most Frequently Used Workloads workspace

We sort the table view on the Ten Most Frequently Used Workspaces workspace from Figure 5-75, in descending order by Number of Occurrences. We see that method `debugOut` is being called excessively by a number of business functions. We can pass this information to the application support team to determine why this debug method is being called so frequently.

5.4.9 Example 11 - Detecting multiple concurrent problems

In this example we constructed a scenario in which three of the preceding problems are happening concurrently:

- ▶ Example 1: A specific user is experiencing slow response while other users work well.

- ▶ Example 4: There is a problem in CICS Transaction Server that is impacting WebSphere.
- ▶ Example 7: Transaction hang or time-out, in this case due to a problem in WebSphere MQ.

In fact, any combination of the preceding examples could have been run concurrently, since OMEGAMON XE's event manager percolates alerts from all agents running on all platforms to the navigator tree view.

Procedure

1. We receive an alert on the ebusiness navigator tree. Position the mouse pointer over the alert (red triangle icon) to show details.

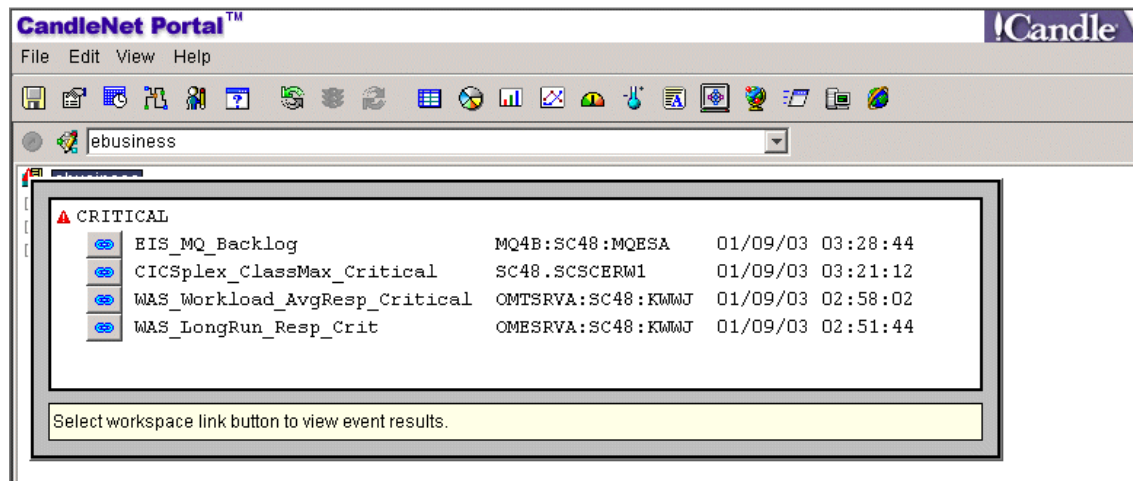


Figure 5-76 Alerts on the ebusiness navigator tree

Figure 5-76 shows that there are four concurrent alerts:

- Alert EIS_MQ_Backlog indicates that the MQSeries queues for application EIS have a backlog.
- Alert CICSplex_ClassMax_critical indicates that CICS region SCSCERW1 that processes the CICS TG requests for application EIS is experiencing CICS task-related issues.
- Alert WAS_Workload_AvgResp_Critical for server instance OMTSRVA indicates that one or more workloads have exceeded the threshold for average response time.
- Alert WAS_LongRun_Resp_Crit for server instance OMESRVA indicates that one or more workloads have exceeded the response time threshold for a single invocation.

2. The question of which alert to investigate first will probably depend on the priority of the business applications. For example, customer-facing applications will probably have a higher priority than internal applications. In this case, you may know that the most critical applications depend on CICS.

Another determining factor is how many users are affected. For example, the WAS_Workload_Avg_Resp_Crit indicates that the average response time for all users is exceeding the threshold. This is probably more serious than the WAS_LongRun_Resp_Crit alert, which indicates that specific users have exceeded the threshold.
3. You can click the link button for any of these alerts to see the Current Situation Values table, which will give you more details. For example, the Current Situation Values Table for alert WAS_Workload_Avg_Resp_Crit will display how many workloads have exceeded the average response time threshold, and their response times. Refer to Figure 5-38 on page 196 for an example.
4. Clicking the link button for a specific alert will also expand the navigator tree to reveal the workspace that contains the relevant performance metrics for that alert. For example, the WAS_Workload_AvgResp_Crit alert automatically expands the navigator tree to the All Workloads workspace for the relevant application server instance.
5. You can continue problem diagnosis as described in the preceding examples.

Example 11 - Advanced procedure

An alternative to the previous procedure is to use the custom business view for the PRR application; refer to “Customized business view” on page 172. With this approach we can proactively monitor all the key components of the PRR application from one workspace.

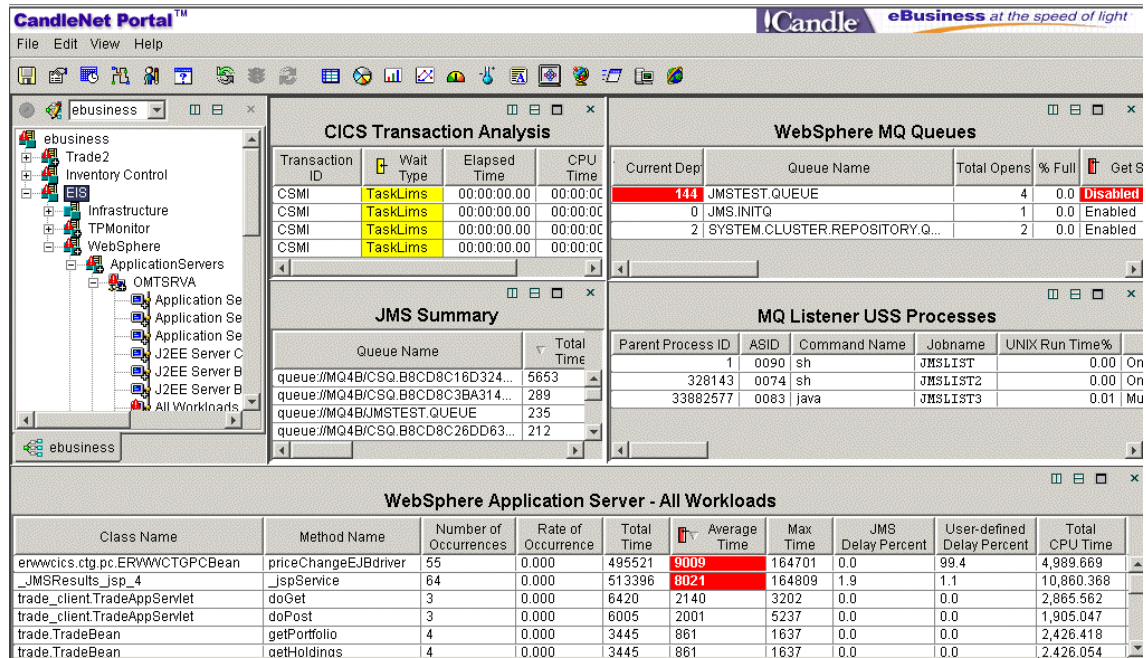


Figure 5-77 Custom business view

Figure 5-77 shows the custom business view that we configured for application PRR. We can see the following alerts related to the PRR application:

- The WebSphere MQ Queues view has an alert that the depth of the trigger queue used by the PRR application has exceeded our threshold of three. This view also has an alert that the queue is Get disabled.
- The CICS Transaction Analysis view has an alert that multiple CSMI transactions are waiting due to a CICS transaction class limit.
- The WebSphere Application Server - All Workloads view indicates that two workloads have average response times greater than 3 seconds.



WebSphere Studio Application Monitor

In this chapter we describe what WebSphere Studio Application Monitor (WSAM) is, how it works, and how it can be used to investigate sample performance problems that we devised for our tests.

6.1 What WebSphere Studio Application Monitor is

WebSphere Studio Application Monitor (WSAM) is the z/OS member of an integrated family of performance management products. This product family supports WebSphere Application Server on all the platforms on which it runs.

WSAM provides visibility into applications deployed in WebSphere. To function, there is no need to modify application byte code or understand application/source code in order to monitor applications. This means that installation takes days rather than weeks.

WSAM helps operations improve service levels by providing operators the ability to monitor and optimize performance, as well as diagnose and fix problems in development, quality assurance, and production environments.

Figure 6-1 provides a solution blueprint of the WSAM product, which manages your application from the inside out. Currently, WebSphere Studio Application Monitor fully monitors tiers 1 and 2 (Java and J2EE), and part of tier 3 (J2EE API).

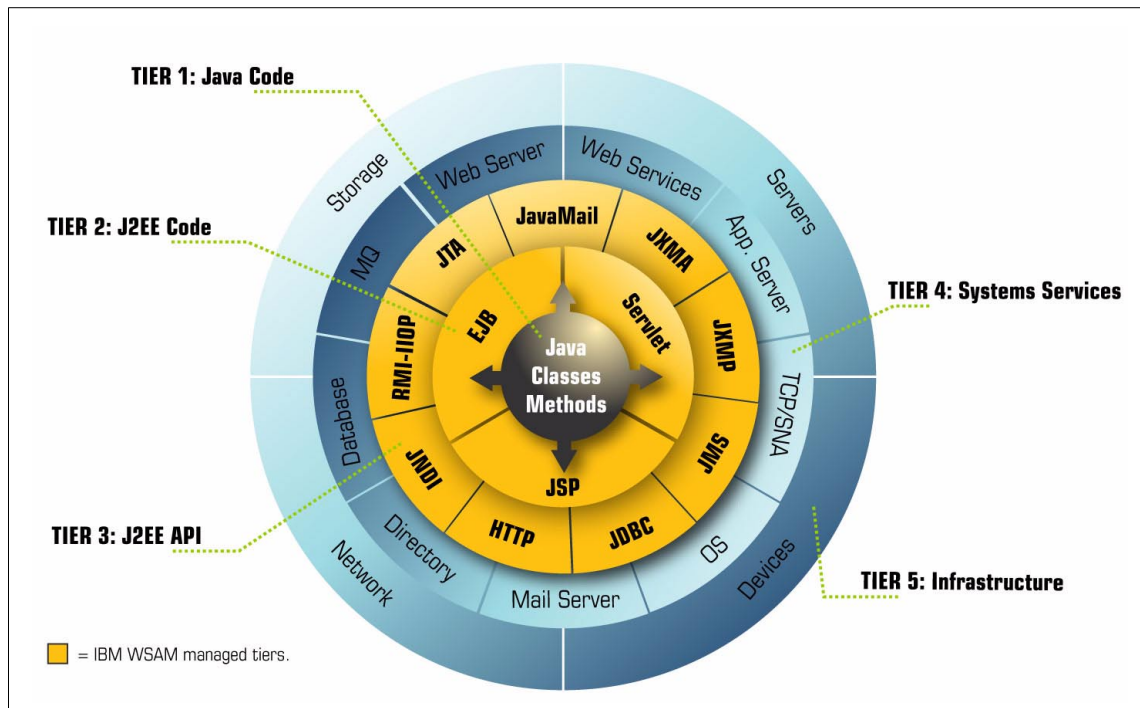


Figure 6-1 WebSphere Studio Application Monitor solution blueprint

WebSphere Studio Application Monitor perspective

WSAM has an application-centric point of view into your enterprise. It lets the operator look at enterprise activity from the inside out, from individual Java class and method calls through J2EE Application Server utilization. WSAM gives you visibility into the WebSphere “black box” and provides the information necessary for managing today’s complex business environments.

In addition, WebSphere Studio Application Monitor is designed with operations in mind. Generally, the user is not expected to have development resources or knowledge to be productive with the tool. However, support from the application development team would provide additional insight for problem determination and performance management.

WebSphere Studio Application Monitor features

WSAM offers operators the following useful features for troubleshooting and operations planning:

- ▶ Drill-down methodology
Progress from a high-level view of an enterprise down to thread-level detail, as part of real-time problem determination or as part of performance analysis and reporting of historical data.
- ▶ Continuously monitor and archive availability and activity information
Determine what is normal and what is exceptional resource use.
- ▶ Configurable Monitoring Level
Limit the impact of data collection on monitored servers by specifying configurations by default, according to a schedule, or on-the-fly.
- ▶ Automated monitoring
Define specific conditions (traps) which, when encountered, trigger logging and/or e-mail delivery of alerts.
- ▶ In-Flight Request Search
Identify and take actions on specific problem threads.
- ▶ Customizable security
Regulate access to WebSphere Studio Application Monitor with user IDs and passwords. Regulate access to monitoring functions based on either predefined or custom roles (Administrator, Operator and User), applied on a user-by-user basis.
- ▶ Compare runtime environments
Define logical groups of servers, and compare configurations and installed binaries on “identical” systems.

- ▶ Performance analysis and reporting
Define and run reports, based on historic data, on virtually any aspect of enterprise use. Reports can detail individual method, Request or SQL calls, WebSphere resource use, or server availability. The scope of reports can include individual servers, server groups, or the entire server farm, and can contrast results against benchmark data sets.

6.2 How WebSphere Studio Application Monitor works

In this section, we provide an overview of WSAM, and describe each of its three major components.

6.2.1 WebSphere Studio Application Monitor architecture

The WSAM architecture consists of three main parts: *Data Collectors*, the *Application Monitor*, and the *Monitoring Console*.

Data Collectors

Data Collectors are installed and attached to each WebSphere instance you want to monitor.

Application Monitor

The Application Monitor is an independent set of services that aggregates the data provided by Data Collectors, and makes sense of it.

Monitoring Console

The Monitoring Console is the client application that allows you to interact with the Application Monitor (and your Data Collectors). The Monitoring Console is a Web-based application, so it can be accessed using any HTML browser.

Figure 6-2 on page 229 provides an overview of how the IBM WebSphere Studio Application Monitor works.

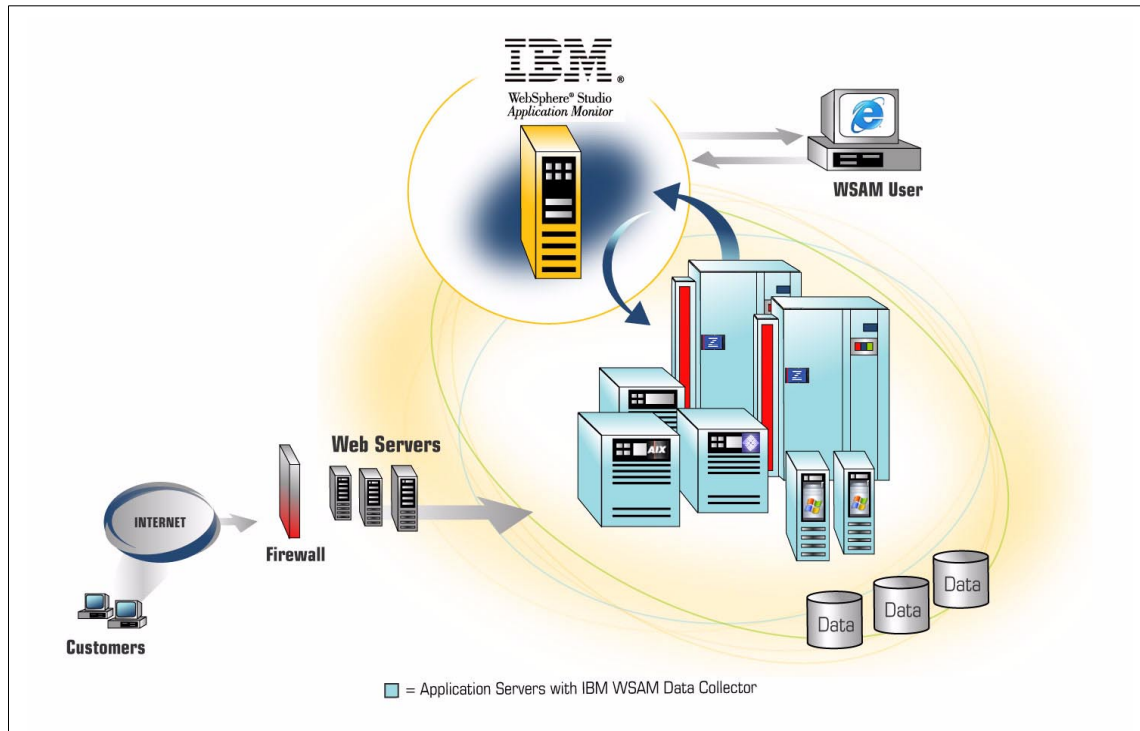


Figure 6-2 IBM WebSphere Studio Application Monitor

WSAM is installed on machines external to the machines running your applications. WSAM monitors your applications via Data Collectors that communicate with the Application Monitor. Each user uses a Web browser client to access WSAM.

6.2.2 WebSphere Studio Application Monitor data collection

In this section, we describe the data collector, which is the component that resides on all devices being managed.

The lifestyle of WSAM data collectors

The core of WebSphere Studio Application Monitor data gathering capability is the WSAM data collector that runs in *each server region*. Although each region has its own copy of the WSAM data collector, all the data collectors within a server instance are treated as one. This means that whatever you configure or select for monitoring on the Application Monitor automatically applies to all the server regions in the WebSphere server instance you selected.

The WSAM data collector does not require any modification to application code. It does not peek inside the classes and methods used in the application; rather, it operates at the container, EJB, servlet, and JSP level.

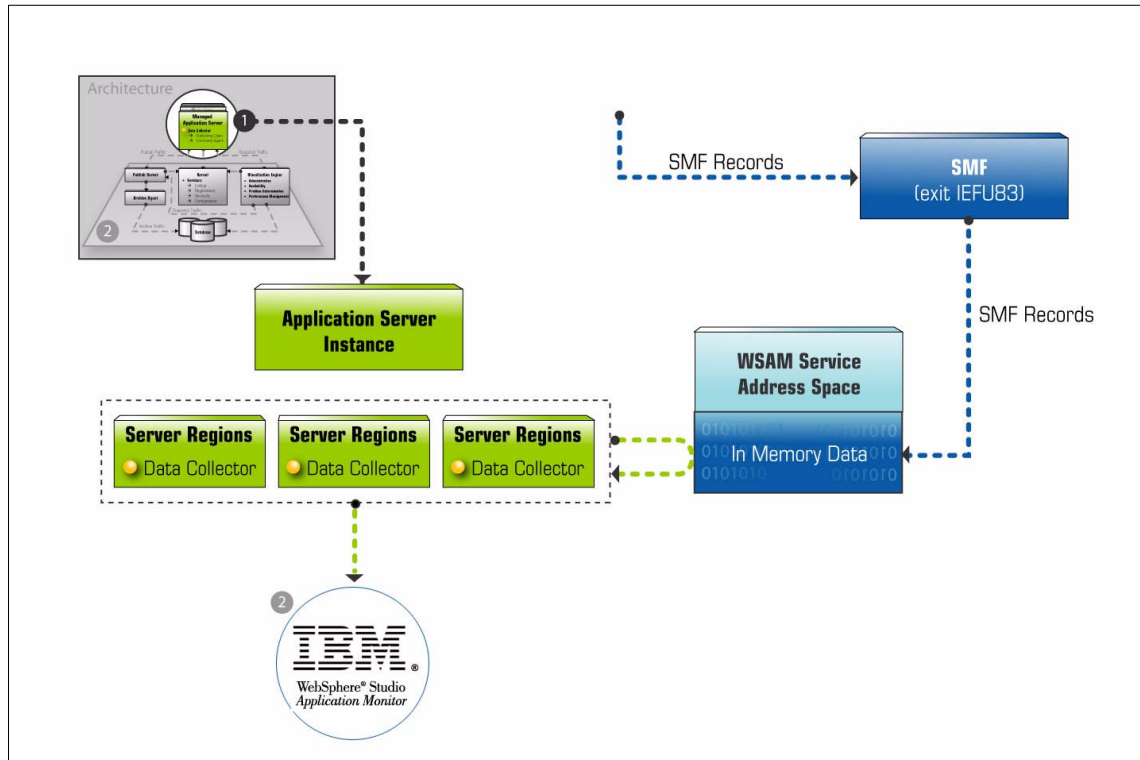


Figure 6-3 WSAM data collector architecture

WSAM data collector configuration

WSAM data collectors can be configured to run in one of three modes:

- ▶ Production
- ▶ Problem Determination
- ▶ Profiling

The Production mode has the least overhead on the system, yet provides most of the availability data for the servers and the system resources used by those servers. Only the most detailed information, such as method traces, is not monitored in Production mode.

The Problem Determination mode is the default. It supplies the data available in Production mode, plus (in the distributed Cyanea/One products) additional problem determination functions. In the current release of IBM WebSphere

Studio Application Monitor on z/OS, the Problem Determination and Production modes provide exactly the same information.

The Profiling mode uses more resources, but it allows you to monitor down to the level of methods and SQL calls. If you need Profiling mode to diagnose a particular problem, you can turn it on and off dynamically for the server or the instance you are looking at.

Furthermore, you can assign configurations to WSAM data collectors by default, according to a schedule, or by explicitly and immediately setting the data collection mode through the **Performance Management -> Monitoring on Demand** section of the WSAM Monitor Console.

Service address space

The WebSphere Studio Application Monitor service address space collects data from SMF type 120 records. The records are viewed by an SMF exit (IEFU83), which extracts the data and hands it to the service address space. Here it is stored in memory until it is either requested by the monitoring console, or a predefined time period has elapsed.

Note: Collecting data from SMF is optional. You will be able to collect and monitor most data relating to the WebSphere servers even if SMF is not collecting type 120 records.

To collect the appropriate SMF data, you need to ensure that the WebSphere servers have been configured to write Server Interval SMF records and Container Interval SMF records. This is done via the SMEUI.

Currently, the service address space only collects and forwards SMF data. Additional functions may be added in future releases.

6.2.3 WSAM Application Monitor

The WSAM Application Monitor is the core of WebSphere Studio Application Monitor. In this section, we describe its implementation and its interactions with other components, in particular with respect to the WSAM data collector.

WSAM Application Monitor implementation

The WSAM Application Monitor runs under AIX or Linux. It runs as a Java service on those platforms, and requires both WebSphere and DB2 products to be installed. See Figure 6-4 on page 232 for an overview of its structure.

The WSAM Application Monitor's use of WebSphere is simply to allow remote access to the Monitor from Web browsers. The WSAM Application Monitor itself is pure Java.

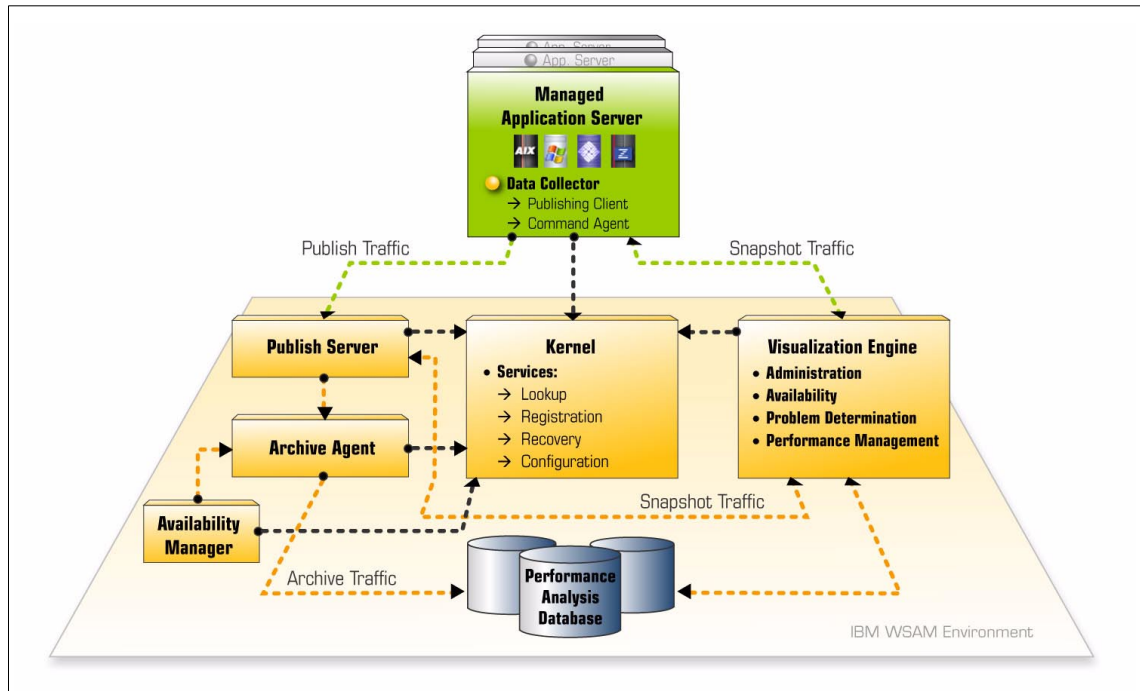


Figure 6-4 WSAM Application Monitor

The WSAM Application Monitor is designed to run on distributed platforms for scalability and availability, although in our tests we ran everything on a single AIX server. WSAM data collectors are configured with two IP addresses each (both the same in our tests), so that two separate kernels may service them. Coordination among the distributed components of WebSphere Studio Application Monitor is accomplished by means of J2EE communications.

Data traffic

Two types of traffic may be distinguished flowing from data collectors to the Application Monitor

Publish Traffic

Publish Traffic includes three types of data, which differ in use and in the mechanism used to collect it: *Application Activity* data, *Server Availability* data and *Application Server Availability* data.

Publish Traffic is archived for subsequent retrieval on demand, as described in “Data archival” on page 233. In addition, the WSAM Monitor Console periodically refreshes its display of Server Availability data (in the Application Overview section) to keep up-to-date with the Publish Traffic.

Snapshot Traffic

Snapshot Traffic is gathered only when you request it; you can request Snapshot Traffic data and view it using the WSAM Monitor Console (in the In-Flight Request Search section).

Snapshot traffic comes from WebSphere via the WSAM data collector. Snapshot Traffic includes information about active threads. This includes both summary information as well as stack traces at the method level.

Data archival

Data from Publish Traffic ends up in the DB2 database of performance data.

The amount of data stored in the DB2 performance database can be very large, especially if Profiling mode is used (see “WSAM data collector configuration” on page 230). Since the database is intended mainly for historical reports, you usually only need to archive a small proportion of your performance data.

If a particular problem is under investigation, you can increase the sampling rate (up to 100% if need be) via the **Administration -> System Properties** section of the WSAM Monitor Console. For the majority of the examples described in 6.5, “Running the examples” on page 237, we used a 1% sampling rate.

6.2.4 WSAM Monitor Console

The WSAM Monitor Console is the component of IBM WebSphere Studio Application Monitor that an operator uses to access all product functionality.

The WSAM Monitor Console is an HTML-based thin client application that runs on IE 5 or later, or Netscape 7 or later.

6.3 Performance methodology

This section describes the philosophy and process of WebSphere Studio Application Monitor performance methodology.

WSAM problem-solving philosophy

WebSphere Studio Application Monitor approaches the detection of performance problems from an operator’s point of view. Very often, operators do not have direct support from developers and yet are given applications to deploy and run. Therefore, WSAM provides a methodology for performance analysis which does not rely upon the availability of developers or source code.

First, we discuss the issue of isolating your performance problems. We present a drill-down methodology that allows operators to start with a real-time view of the data center, and allows them to methodically drill down to the cause of the problem. This does not require familiarity of the deployed application code.

The design of IBM WebSphere Studio Application Monitor lends itself to a top-down approach to performance management, as shown in Figure 6-5.

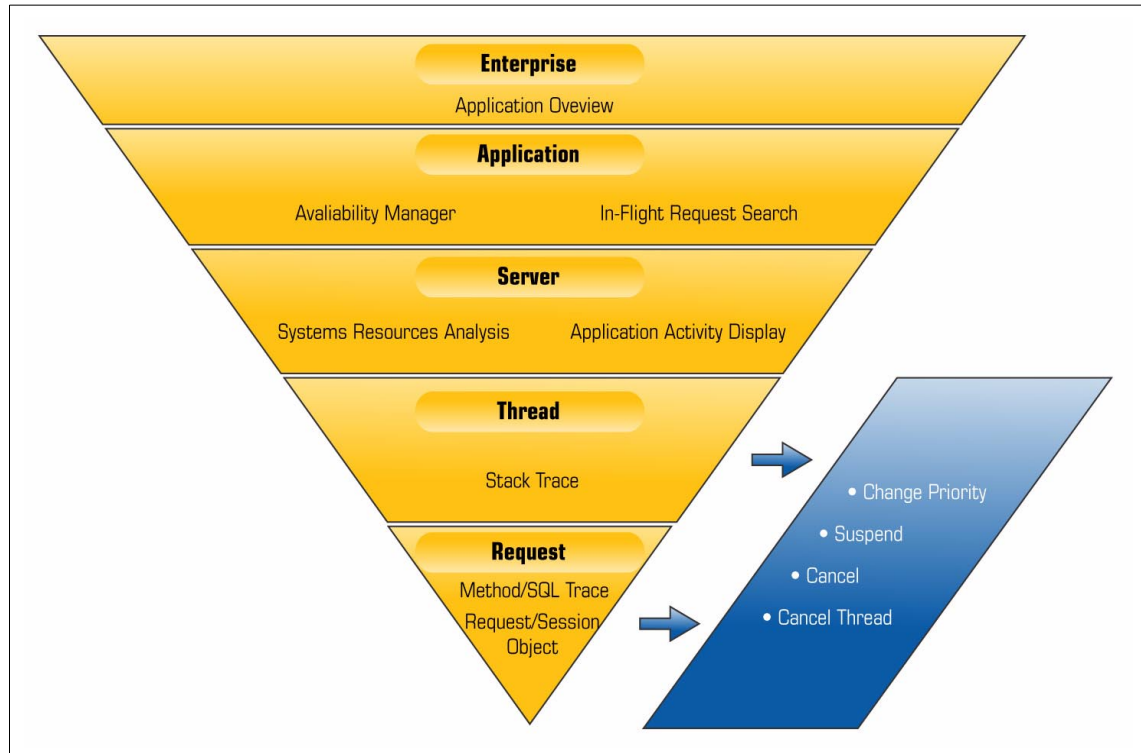


Figure 6-5 WSAM monitoring philosophy. The eiderdown starts from the Enterprise and progresses to Application, then Server, then Thread, then Request

In addition, you can analyze performance of transactions after they have completed. In the Performance Analysis and Reporting section, you can analyze performance and drill down into the problem.

WebSphere Studio Application Monitor understands that one size does not fit all. Since there are trade-offs between the amount of data collected and system performance, you can configure the application monitor to capture the appropriate amount of data, on a server-by-server basis. This configuration can be permanent, or can vary based on a schedule, and can always be overridden

to diagnose a particularly interesting problem; see “WSAM data collector configuration” on page 230 for more detail.

WSAM problem-solving methodology

The usual starting point for solving a problem with an Enterprise deployment, is the Application Overview section. This display shows you, in graphical form, how many servers are active, the volume of throughput, and an indication of response time for your entire server farm.

Drill-down approach

From the Application Overview section, you can proceed directly to one of three more detailed views of any particular server group: In-Flight Request Search, Server Availability Detail, or System Resource Comparison:

- ▶ **In-Flight Request Search**
Lets you identify and act upon resident threads. Useful if you expect that the problem transaction has not completed.
- ▶ **Server Availability Detail**
Provides automatically updated server availability metrics for each server in the group, like absolute and delta volume, memory, and CPU use. Useful to compare servers within a group in more detail, and to let WSAM aid you in identifying poor performance metrics with visual cues.
- ▶ **System Resource Comparison**
Compares environments and deployed binaries among servers in a group. Useful for investigating issues where one server among a group is behaving erratically.

At any level of investigation, you are presented with links to the relevant next-step features. For example, from the Server Availability Detail page, once you identify a particular problem server, you can look at its resident threads in the Application Activity Display, look at its J2EE resource use in the System Resources Overview, or proceed to the System Resource Comparison.

Continuous monitoring: traps

WebSphere Studio Application Monitor lets you monitor your servers automatically, allowing you to catch a particular behavior or event at the time it occurs, through Trap & Alert Management.

You can define traps to look for a specific event by name and type (HTTP or SQL), for a specific application behavior or metric, or for WebSphere performance metrics that reach or surpass defined thresholds.

WSAM lets you define the actions to take when a trap is triggered, which includes logging and e-mail. You can define the number of times the trap condition must be met before WSAM executes the trigger.

Analysis-based methodology

In addition to the “real-time” use, you can define a variety of reports and charts from the archived data; for example, throughput by application group against time of day.

Reports are useful both for capacity planning as well as problem determination.

6.4 ITSO configuration

At ITSO, we ran the application monitor on a single AIX machine. The data collectors were installed on the WSTSRV and WSESRV server instances on all three LPARs. We switched on SMF type 120 recording and ran the service address space in all three LPARs.

Data collectors and service address space

Data Collector Classname Exclude List: filter out sqlj, which is used as part of the eITSO application

Application monitor

Installed on a single AIX server F50 2-way 322 MHz machine with 1 GB of memory. This was a little under-powered for our purposes.

We configured the default Sampling Frequency as 1%. Therefore, in normal operation only one percent of all transactions were logged by the Data Collectors to the application monitor.

The application monitor was used with the default user, the administrator, which had Administrator permissions for the product. In production you might well restrict various users’ powers, but since this does not add to our discussions of performance monitoring, we adopted the simplest solution.

Two server groups were set up in the monitoring console. The group names were eITSO Sample Application (for the eITSO server) and Trade2 Application (for the Trade2 and PRR server).

Monitoring console

The monitoring console is a Web-based client. Therefore, it can run on any compliant browser. In our test lab, we used IE 6.0 on Windows 2000 as the client

workstation. We also did some work using Mozilla Version 6, although the captured screen shots are all IE for consistency.

Features used

WebSphere Studio Application Monitor core product only. No additional product add-ons were used.

6.5 Running the examples

Most of the panels seen on the WSAM console are extremely detailed and do not lend themselves well to reproduction in a book such as this. Therefore, we have cropped the screen shot images in order to display clearly the information most relevant to the discussions.

6.5.1 Example 1

In Example 1, a particular user experiences a delay with his transaction, which normally runs fine for other users. We want to isolate the offending transactions.

Methodology

Our methodology is as follows:

1. Since this is not an in-flight transaction, we start with the Performance Analysis and Reporting section of WSAM to isolate time periods in which requests took a long time to execute.
2. We drill down on the offending time intervals by looking at the requests executed within them, and then drill down further to the underlying method trace. This provides details we can pass back to the developers for further analysis.

Procedure

We start with a general search for all requests within the half-hour interval in which the problem occurred, on a minute-by-minute basis, by performing a Request Analysis. This is accomplished in our ITSO lab setup as follows:

To search for all requests in the half-hour interval in which the problem occurred, on a minute-by-minute basis, do the following:

1. Navigate to **Performance Management -> Performance Analysis & Reporting**.

This brings up the Performance Analysis and Reporting page.

2. Click **Define Report** in the left side navigation to create a new report.

This brings up the Server and Report Type Selection page.

3. Enter the following information in the Server and Report Type Selection section:

- Group: eITS0 Sample Application
- Server: All Servers
- Report Type: Request Analysis

Click **Next>**. This brings up the Report Filtering Options page.

4. Enter the following information in the Report Filtering Options section:

- Metric: Response Time
- Request Type: ALL
- Request Name: <leave blank>

Click **Next>**. This brings up the Data Set Parameters page.

5. Enter the following information in the indicated sections:

- Start Date: 11/13/02 8:30 PM
- End Date: 11/13/02 9:00 PM
- Contrast Options: None
- Data Grouping: Minute of the Hour

Click **Finish**. The results of the Trace Report appear in Figure 6-6 on page 239.

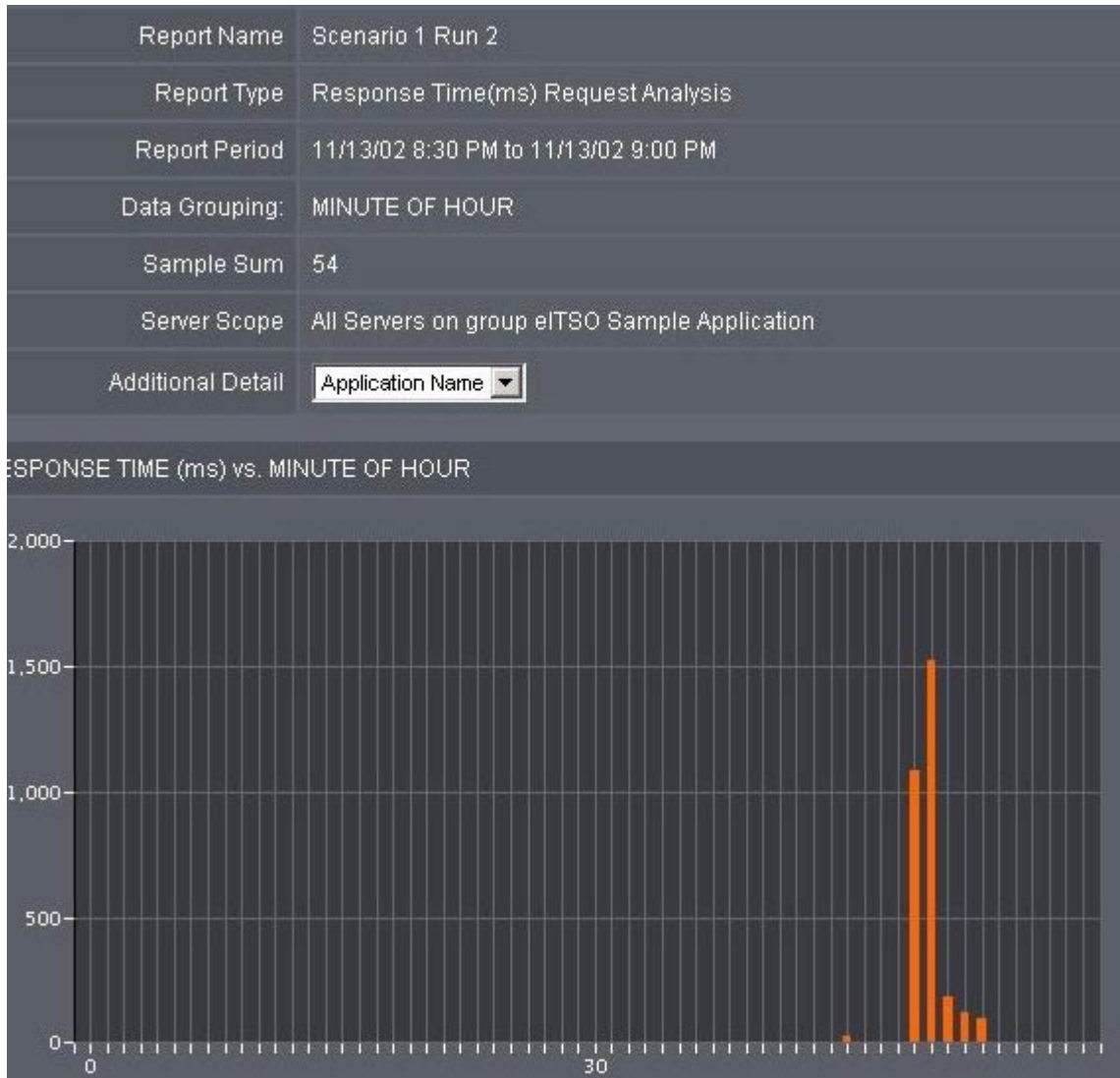


Figure 6-6 Trace Report: Shows the response time trend of requests during the specified period

The graph in Figure 6-6 shows two dramatic spikes in the average response time for requests. Next, we want to find out more information about the requests in the most offensive time period.

To find out more information about the requests in the time period with the highest response time, do the following:

1. Select **Application Name** in the Additional Detail field of the Report Properties section (this should be the default selection).
 2. Click the bar that has the highest response time.
- Figure 6-7 on page 241 shows the resulting decomposition of requests for the selected period.

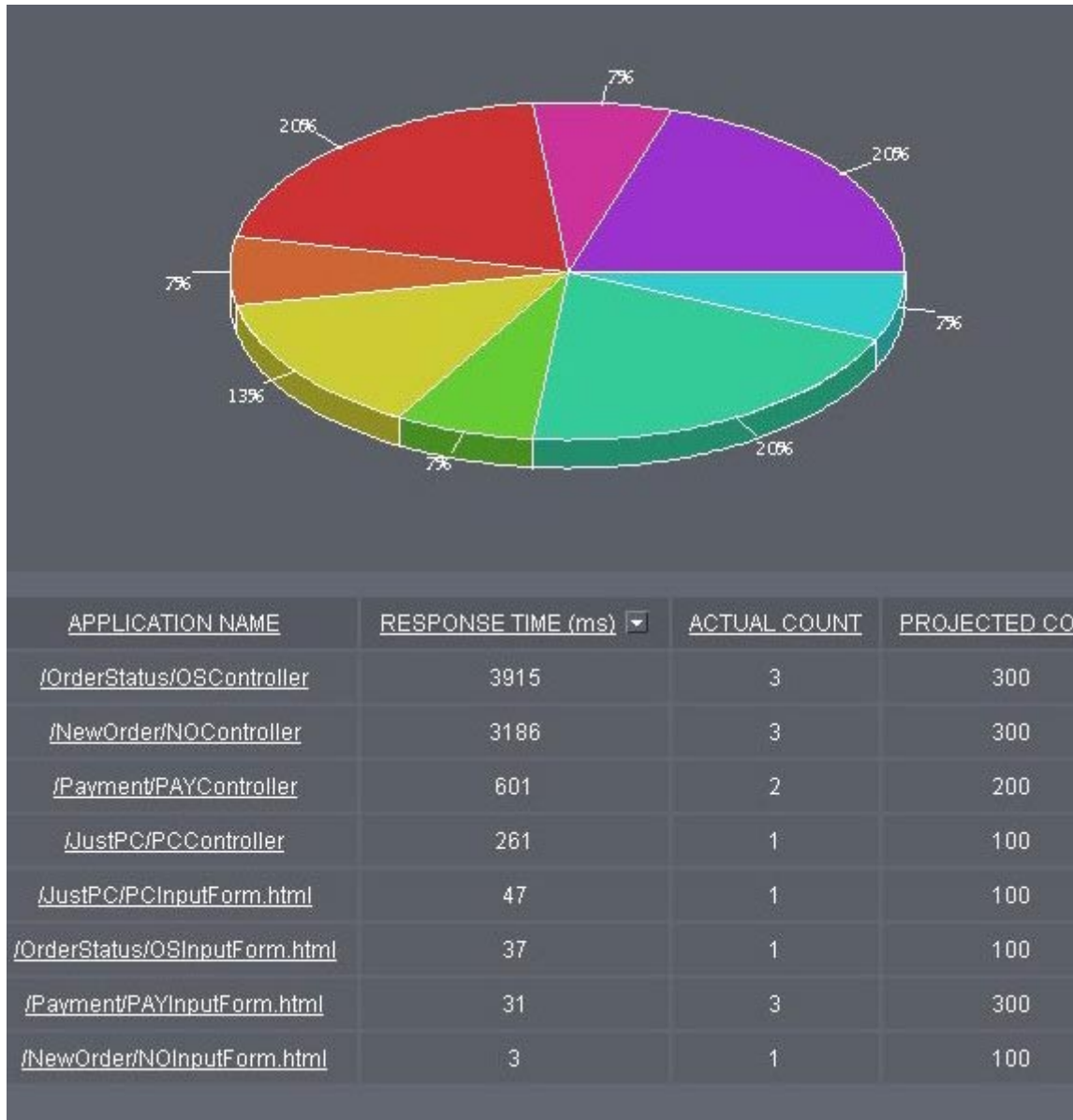


Figure 6-7 Decomposition Report: Shows the Web request composition of a selected minute.

Notice that the projected count is different from the actual count. This means that the sampling frequency was not 100%.

- To sort the results by descending response time, click the heading **RESPONSE TIME (ms)** twice.

We find that the request for the URI `/OrderStatus/OSController` was the one that, on average, took the longest to complete.

- To see why this was the case, click the **/OrderStatus/OSController** link.

These results are shown in Figure 6-8.

REPORT PROPERTIES				
Report Name	Scenario 1 Run 2			
Report detail on	MINUTE OF HOUR : 50			
Report decomposed by	Application Name on MINUTE OF HOUR : 50			
Report Type	Response Time(ms) Request Analysis			
Report Period	11/13/02 8:30 PM to 11/13/02 9:00 PM			
Server Scope	All Servers on group eITSO Sample Application			

<u>REQUEST NAME</u>	<u>REQUEST TYPE</u>	<u>RESPONSE TIME (ms)</u> ▾	<u>CPU TIME (ms)</u>	<u>SERV</u>
/OrderStatus/OSController?customerLastName=MIN&command=Manua	Servlet	8313	7384.129	WTSCPLX1.
/OrderStatus/OSController?customerLastName=&command=Autogen&	Servlet	1995	298.078	WTSCPLX1.
/OrderStatus/OSController?customerLastName=&command=Autogen&	Servlet	1437	152.950	WTSCPLX1.

Figure 6-8 Request Detail: Shows the individual requests that were captured during the one-minute interval of highest response time.

- Once again, click the heading **RESPONSE TIME (ms)** twice to sort the requests by descending response time.

We see that one request was dramatically slower than the rest. Furthermore, this request has different query parameters than the other requests.

6. To determine the cause of the slowdown, we drill down on this request by clicking its name:
/OrderStatus/OSController?customerLastName=MIN&command=manua
 The resulting method trace is shown in Figure 6-9.

an.findCustomerByLastName	Date/Time	Nov 13, 2002 8:50:47 PM	Elapsed Time	20 ms	
CBIVP.customer					
ement.executeQuery	Date/Time	Nov 13, 2002 8:50:47 PM	Elapsed Time	25 ms	C
FROM CBIVP.customer					
ment.executeQuery	Date/Time	Nov 13, 2002 8:50:47 PM	Elapsed Time	25 ms	C
Bean.findByPrimaryKey	Date/Time	Nov 13, 2002 8:50:55 PM	Elapsed Time	7835 ms	CPU Time
findByPrimaryKey	Date/Time	Nov 13, 2002 8:50:55 PM	Elapsed Time	7836 ms	
d FROM CBIVP.customer WHERE c_id =					
checkConnection	Date/Time	Nov 13, 2002 8:50:55 PM	Elapsed Time	7836 ms	CP
an.makeConnection	Date/Time	Nov 13, 2002 8:50:55 PM	Elapsed Time	7836 ms	

Figure 6-9 Method Trace: Shows the individual methods executed in the selected request.

This page shows the individual methods executed in our offending request. We see that both the elapsed time and the CPU time jump dramatically between the call to the database and the subsequent call to `findByPrimaryKey`.

This specific behavior can be taken back to the development organization for further analysis, so they can determine the cause of the problem and how to address it.

6.5.2 Example 3

Example 3 is an example of a memory leak. We can use WebSphere Studio Application Monitor to observe memory usage in real time.

Methodology

Our methodology for identifying memory is to observe server availability metrics in real time, including JVM Memory use.

Procedure

We observe the JVM Memory use in real time. This is facilitated by setting an availability threshold for JVM Memory use, which allows WSAM to automatically highlight cases where use passes the threshold.

To observe the current memory usage, do the following:

1. Navigate to **Availability** -> **Server Availability Detail**.

The Application Overview page appears.

2. Click the arrow to the right of the All Servers entry in the Server Tree section on the left side of the page to view the availability details of all servers.
3. Observe the values in the JVM Memory Usage (mb) column.

The information on the Server Availability Detail page refreshes periodically.

If there is a memory leak, the value of the JVM Memory Usage (mb) column may fluctuate from page refresh to page refresh. But, on the average, the JVM Memory Usage value will increase over time.

To aid detection of excessive JVM Memory Usage, configure the Server Availability Detail to automatically indicate when JVM Memory Usage surpasses a threshold. This will help you quickly spot applications that are currently using more memory than others. It is normal to see high memory usage periodically as the heap fills before each garbage collection, but if high memory usage becomes increasingly frequent over a sustained period of time, this might suggest the presence of a memory leak.

To define a threshold for acceptable JVM Memory usage, do the following:

1. Click **Configuration** in the Refresh Settings section.

The Server Availability Configuration window pops up.

2. In the drop-down next to JVM Memory Usage, select > (greater than).
3. In the field next to the right of the drop-down, enter the threshold for acceptable JVM Memory usage, in mb. In our case, enter 100.
4. Click **Save**.

The pop-up window closes.

Once you have set a Server Availability Detail threshold, WSAM automatically highlights, in yellow, the names of any servers on the Server Availability Detail page that pass the configured thresholds.

Figure 6-10 shows the Server Availability Detail page in which three servers are experiencing high JVM Memory usage. If this usage continues or increases, it is possible that a memory leak exists.

SERVER DETAIL								
Name ▲		Status	Platform	Volume Δ	Total Volume	Total CPU %	JVM Memory Usage (mb)	Group
<input checked="" type="checkbox"/> WTSCPLX1 WSESRVA 1010 (L1)	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	Available	z/OS	0	5	50.00	131	eITSO S Applic
<input checked="" type="checkbox"/> WTSCPLX1 WSESRVB 29 (L1)	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	Available	z/OS	34	1,196	8.00	120	eITSO S Applic
<input checked="" type="checkbox"/> WTSCPLX1 WSESRVB 87 (L1)	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	Available	z/OS	0	0	6.00	14	eITSO S Applic
<input checked="" type="checkbox"/> WTSCPLX1 WSESRVC 102 (L1)	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	Available	z/OS	83	1,583	54.00	137	eITSO S Applic
<input checked="" type="checkbox"/> WTSCPLX1 WSESRVC 103 (L1)	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	Available	z/OS	0	17	3.00	18	eITSO S Applic
<input checked="" type="checkbox"/> Clear All								

Figure 6-10 Server Availability Detail: Shows detail statistics related to server availability, including JVM Memory Usage.

Follow-up

Once you have found a server with a memory leak, the next step is to locate the source of the problem.

If the memory leak is confined to a single server out of a server group, one possible explanation is that the version of the application installed on the problem server is not up-to-date. WebSphere Studio Application Monitor provides a way to compare installed binaries among servers in a server group, through the *Software Consistency Check: Installed Binary Comparison*.

6.5.3 Example 4

In Example 4, users are complaining that response time is increasing—what used to take around one second is taking three or more. We know that their

transactions use the Trade2 server, and we know that the problem has been around for some time, so we have archived a fair amount of data.

Methodology

Our methodology is as follows:

1. Since we have archived data describing this problem, we start with the Performance Analysis and Reporting section of WSAM to isolate time periods in which requests took a long time to execute.
2. We drill down on the offending time intervals by looking at the requests executed within them, and then drill down further to the underlying method trace.
3. Using the System Resources Overview, look at EJB use, and drill down to see the underlying method trace of the relevant EJBs.

This provides details we can pass back to the developers for further analysis.

Procedure

We decide to run a Request Analysis report on the Trade2 server. This is done from the Performance Analysis and Reporting section of WSAM.

To run a Request Analysis on the Trade2 server, do the following:

1. Navigate to **Performance Management -> Performance Analysis & Reporting**.
This brings up the Performance Analysis and Reporting page.
2. Click **Define Report** in the left side navigation to create a new report.
This brings up the Server and Report Type Selection page.
3. Enter the following information in the Server and Report Type Selection section:
 - Group: Trade2 Application
 - Server: All Servers
 - Report Type: Request AnalysisClick **Next>**. This brings up the Report Filtering Options page.
4. Enter the following information in the Report Filtering Options section:
 - Metric: Response Time
 - Request Type: ALL
 - Request Name: <leave blank>Click **Next>**. This brings up the Data Set Parameters page.
5. Enter the following information in the indicated sections:

- Start Date: 11/13/02 8:30 PM
- End Date: 11/13/02 9:00 PM
- Contrast Options: None
- Data Grouping: Minute of the Hour

Click **Finish**. The results of the Trend Report appear in Figure 6-11 on page 247.

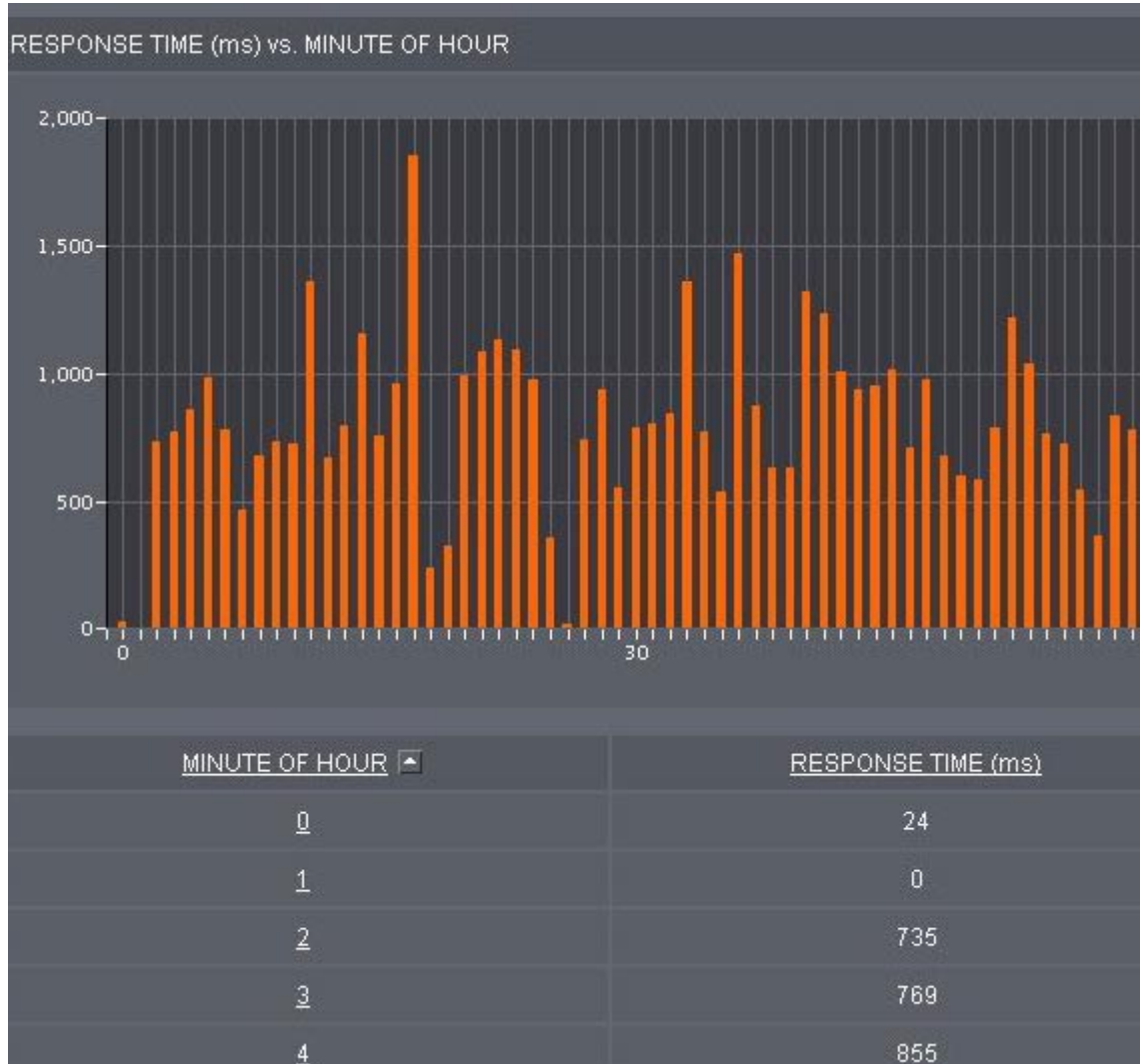


Figure 6-11 Trend Report

Our Trend Report shows that, indeed, the response time *within the WebSphere server region* is often well over a second. Combined with network delay, this may very well explain users' complaints.

We proceed to identify the requests and underlying methods that cause these glacial response times within the WebSphere server region.

To determine which requests and underlying methods have excessive response times, do the following:

1. On the Trend Report, click any of the bars with offending response times.

The resulting decomposition of requests for the selected period appears in Figure 6-12.

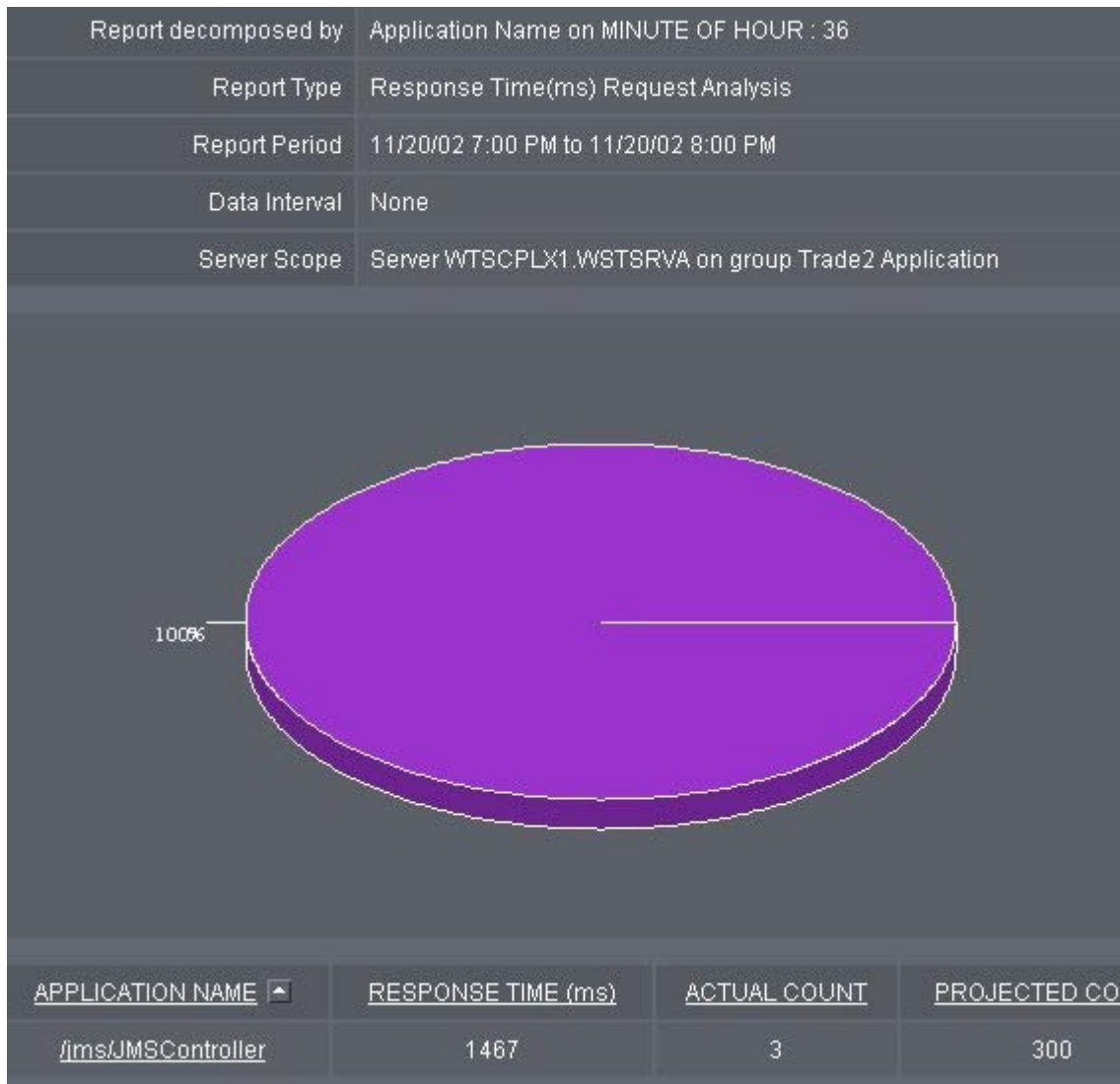


Figure 6-12 Decomposition report

Our Decomposition Report shows that the sole offending request is for the `/jms/JMSController`.

2. To see a breakdown of the requests that comprise the JMSController servlets, click the `/jms/JMSController` link. The results are shown in Figure 6-13.

REQUEST REPORT DETAIL
The Request Report Detail displays a breakdown of the data for the portion of the Decomposition Report

REPORT PROPERTIES	
Report Name	
Report detail on	MINUTE OF HOUR : 36
Report decomposed by	Application Name on MINUTE OF HOUR : 36
Report Type	Response Time(ms) Request Analysis
Report Period	11/20/02 7:00 PM to 11/20/02 8:00 PM
Server Scope	Server WTSCPLX1.WSTSRVA on group Trade2 Application

<u>REQUEST NAME</u>	<u>REQUEST TYPE</u>	<u>RESPONSE TIME (ms)</u>	<u>CPU TIME (ms)</u>	<u>SERVER</u>
/jms/JMSController	Servlet	1571	5.975	WTSCPLX1.WSTSRVA
/jms/JMSController	Servlet	1466	6.214	WTSCPLX1.WSTSRVA
/jms/JMSController	Servlet	1364	5.939	WTSCPLX1.WSTSRVA

Figure 6-13 JMSController servlet detail

The Request Report Detail report shows three servlet instances of the offending `/jms/JMSController` invocation.

- To see the method trace of any one of these instances, click one of the `/jms/JMSController` links.

The resulting method trace is shown in Figure 6-14 on page 251.

veTransactionInfo.equals	Date/Time	Nov 20, 2002 3:16:30 PM	Elapsed Time	291 ms
TransactionInfo.equals	Date/Time	Nov 20, 2002 3:16:30 PM	Elapsed Time	291 ms
hManagerImpl.removeFromControlMap	Date/Time	Nov 20, 2002 3:16:30 PM	Elapsed Time	
ctionImpl.isRollbackOnly	Date/Time	Nov 20, 2002 3:16:30 PM	Elapsed Time	291 ms
onImpl.isRollbackOnly	Date/Time	Nov 20, 2002 3:16:30 PM	Elapsed Time	291 ms
tionImpl.afterCompletion	Date/Time	Nov 20, 2002 3:16:31 PM	Elapsed Time	760 ms
ContainerTx.afterCompletion	Date/Time	Nov 20, 2002 3:16:31 PM	Elapsed Time	760 m
er/ContainerTx.persisterAfterCompletion	Date/Time	Nov 20, 2002 3:16:31 PM	Elapsed Time	
/ContainerTx.persisterAfterCompletion	Date/Time	Nov 20, 2002 3:16:31 PM	Elapsed Time	
er/ContainerTx.becomeCommitted	Date/Time	Nov 20, 2002 3:16:31 PM	Elapsed Time	76

Figure 6-14 JMSController method trace

This method trace shows methods with long response times, including `com/ibm/ws390/tx/TransactionImpl.afterCompletion`. Since we know this method calls CICS, we can pass the problem to the CICS team for investigation.

To be thorough, we also inspect EJB use on the Trade2 servers, which is done from the System Resources Overview section.

To see the EJB use on the Trade2 servers, do the following:

1. Navigate to **Performance Management -> System Resources**.

This brings up the System Resources Overview page.

2. Enter the following information in the Choose Server section, on the left side of the page:

- Group: Trade2 Application
- Server: All Servers

The System Resources Overview page refreshes and displays a resource overview for the Trade2 servers.

3. Click the **EJBs** link in the left navigation.

This brings up the EJB Summary page for the Trade2 servers, which is shown in Figure 6-15 on page 252.

EJB SUMMARY			
The EJB Summary page reports information for the enterprise java beans on the selected server.			
SNAPSHOT INFO			
APPLICATION SERVER NAME:	WTSCPLX1.WSTSRVA		
INTERVAL START/END:	Wed Nov 20 14:59:39 EST 2002 - Wed Nov 20 21:48:22 EST 2002		
EJB SUMMARY			
AMC Name	EJB type	Reentrant	Method
RemoteWebContainer::RemoteWebContainer.jar:RemoteWebContainer	2	false	2
PRRs::CICSPRR.jar:ERWWCTGPC	2	false	2
PRRs::PRR_WebApp.jar:PRR_WebApp	2	false	3

Figure 6-15 EJB summary

From the EJB Summary page, we see, among the EJBs used by the Trade2 application, an EJB called PRRS::CICSPRR.jar::ERWWCTGPC. We know this EJB calls CICS, so we wish to investigate it further.

- To see which methods the PRRS::CICSPRR.jar::ERWWCTGPC EJB calls, click **PRRS::CICSPRR.jar::ERWWCTGPC**.

This brings up the EJB Method Summary page, which is shown in Figure 6-16 on page 253.

EJB METHOD SUMMARY				
The EJB Method Summary reports information for the methods of the selected enterprise java bean.				
SNAPSHOT INFO				
APPLICATION SERVER NAME:		WTSCPLX1.WSTSRVA		
RESOURCE:		PRRs::CICSPRR.jar:ERWWCTGPC		
INTERVAL START/END:		Wed Nov 20 14:59:39 EST 2002 - Wed Nov 20 21:48:22 EST 2002		
EJB METHOD SUMMARY				
Method Signature	Invocations	Average Response Time	Max Response Time	Transaction Policy
priceChangeEJBdriver (erwwcics.ctq.pc.PriceChangeInput)	36728	1502	444340	2
create()	36732	0	172	0

Figure 6-16 EJB details

The EJB Method Summary shows that the method `priceChangerEJBdriver (erwwcics.ctq.pc.PriceChangeInput)`, which also calls CICS, shows a response time of over 1.5 seconds.

This level of detail greatly helps the CICS team diagnose the problem.

6.5.4 Example 6

In Example 6, the system runs smoothly with no problems, until one user complains about very slow response time—around five minutes.

Methodology

There are a variety of approaches we can take using IBM WebSphere Studio Application Monitor to locate the problem. In all cases, we locate the request and drill down to get the most specific information possible, including a method trace

of the underlying servlet, EJB, or SQL call. The biggest decision is how to get started. Our options include the following:

- ▶ Run a report using Performance Analysis and Reporting to locate the request.

This path will only work if the user's request was archived as part of normal sampling. Since our database sampling rate is 1% (which is a reasonable figure in a stable production environment), it is unlikely that we will see this one user's request in Request Analysis reports. While this may work, we do not start with this path.

- ▶ Increase the Request Sampling Rate to 100% using System Properties, wait for the user to issue the request, then run a report using Performance Analysis and Reporting.

This option relies on the user reissuing the request, and has the drawback that switching the sampling rate to 100% will fill up the database for the sake of one request. So we do not pursue this path.

- ▶ Set an Application Event trap to find the user's request when it arrives, by using Trap & Alert Management.

This option also relies on the user issuing the request again. Since we think the request may be accessible in recent activity, we do not pursue this path first.

- ▶ Look for the user's request in-flight using Application Activity Display.
- ▶ Look at recent resource use using System Resources Overview.

These two final options differ primarily in that the Application Activity Display panel provides an instantaneous snapshot of the requests being serviced by WebSphere, while System Resources Overview is a picture of recent activity taken from the SMF records. We decide to follow the System Resources Overview path since it provides a broader window into recent activity, and is more likely to provide a result.

Procedure

We start with the System Resources Overview page and use it to locate recently used resources (specifically EJBs), but which are used infrequently, since our problem is not common. We proceed to drill down and generate a method trace of the offending resources.

To use the System Resources Overview path to determine the method causing the request to hang, do the following:

1. Navigate to **Availability -> Server Availability Detail**.

The Server Availability Detail screen appears.

- On the Server Availability Detail screen, in the Server Tree menu, click the arrow to the right of the eITS0 Sample Application server group.

The availability information for the servers in the eITS0 Sample Application server group appears in the Server Detail section, which is shown in Figure 6-17.

REFRESH SETTINGS								
Refresh Interval (sec.)	15	Pause	Refresh	Configuration	Next Refresh (sec)			
SERVER DETAIL								
Name		Status	Platform	Volume	Total Volume	Total CPU %	JVM Memory Usage (mb)	Group
WTSCPLX1 WSES RVA 1017 (L3)	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	Available	z/OS	0	0	64.00	13	eIT Sar Appli
WTSCPLX1 WSES RVA 1022 (L3)	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	Available	z/OS	1,767	101,333	100.00	172	eIT Sar Appli
WTSCPLX1 WSES RVB	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	Unavailable	z/OS	0	0	0.00	0	eIT Sar Appli

Figure 6-17 Server Availability Detail: Shows detail statistics related to server availability across all regions in the server instance.

- To see the System Resource use of servers in the eITS0 Sample Application, click **SR** next to the appropriate server instance name.

Note: Although SR is selected for an individual server region, the data supplied is aggregated across all regions in the server instance.

The System Resources Overview appears, which is shown in Figure 6-18 on page 256.

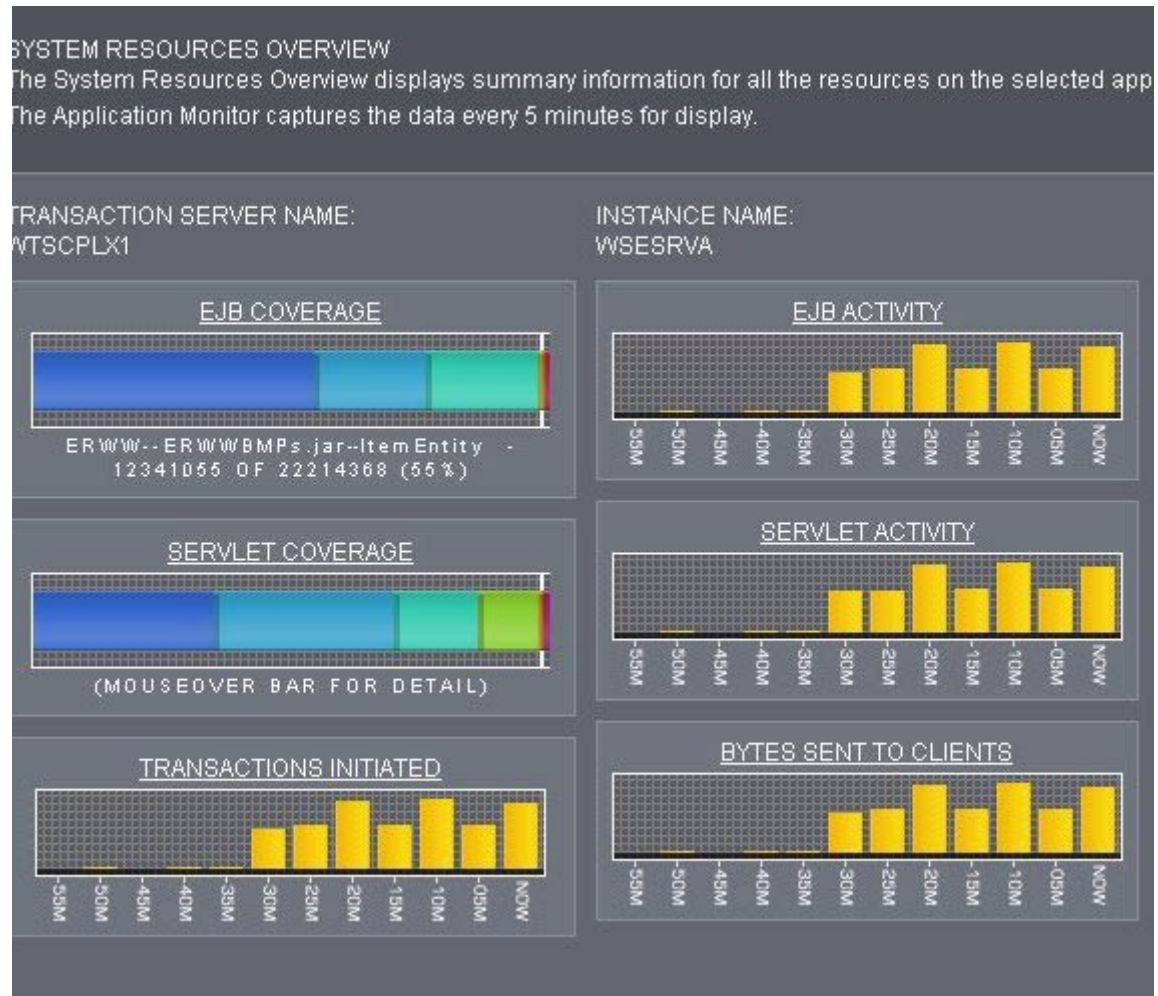


Figure 6-18 System Resources Overview: Shows an Application Server's resource use, including EJB coverage. Mousing over the EJB Coverage graph displays the names of the EJBs used.

The System Resources Overview screen shows, among other things, which EJBs and servlets have been invoked. We suspect that the offending methods lie in a rarely used EJB.

- To identify the names of the EJBs represented in the EJB Coverage box, mouse over each differently colored section of the graph.

Notice that there are some little-used EJBs, which are represented by the short red bar at the right hand end of the EJB Coverage box. Mousing over these seldom-used EJB displays their names.

- To investigate these EJBs further, click the **EJBs** link in the left navigation. The EJB Summary screen appears, which is shown in Figure 6-19.

EJB SUMMARY			
The EJB Summary page reports information for the enterprise java beans on the selected server.			
SNAPSHOT INFO			
APPLICATION SERVER NAME:	WTSCPLX1.WSESVA		
INTERVAL START/END:	Sun Nov 17 23:58:39 EST 2002 - Mon Nov 18 11:10:17 EST 2002		
EJB SUMMARY			
AMC Name	EJB type	Reentrant	Method
ERWW::WebERWWOS_WebApp.jar::WebERWWOS_WebApp	2	false	3
RemoteWebContainer::RemoteWebContainer.jar::RemoteWebContainer	2	false	2
ERWW::ERWWBMPs.jar::OrderEntity	1	false	4
ERWW::OrderStatusSessionJava.jar::OrderStatusSession	2	false	2
ERWW::ERWWBMPs.jar::OrderlineEntity	1	false	6
ERWW::ERWWBMPs.jar::CustomerEntity	1	false	7
ERWW::WebERWWJustPC_WebApp.jar::WebERWWJustPC_WebApp	2	false	3
ERWW::PriceChangeSessionJava.jar::PriceChangeSession	2	false	2
ERWW::ERWWBMPs.jar::ItemEntity	1	false	3

Figure 6-19 EJB Summary

Based on the infrequently used EJBs, we have a good idea of which EJBs are likely to have been invoked during the hanging request. In this case, we look for the `CustomerEntity` bean.

- To see the method summary for this EJB, click the **ERWW::ERWWBMPs.jar::CustomerEntity** link.

The EJB Method Summary screen appears, which is shown in Figure 6-20.

EJB METHOD SUMMARY				
The EJB Method Summary reports information for the methods of the selected enterprise java bean.				
SNAPSHOT INFO				
APPLICATION SERVER NAME:		WTSCPLX1.WSE8RVA		
RESOURCE:		ERWW::ERWWBMPs.jar::CustomerEntity		
INTERVAL START/END:		Sun Nov 17 23:58:39 EST 2002 - Mon Nov 18 11:10:17 EST 2002		
EJB METHOD SUMMARY				
Method Signature	Invocations	Average Response Time	Max Response Time	Transac Policy
<u>findCustomerByLastName</u> (java.lang.String,short,short,boolean)	1	409728	409728	3
<u>getCustomerBalance()</u>	5	0	0	3
<u>findByPrimaryKey</u> (customerEntityPackage.CustomerEntityKey)	4	293379	305053	3
<u>getCustomerId()</u>	10	0	1	3
<u>getCustomerMiddle()</u>	5	0	1	3
<u>getCustomerFirst()</u>	10	0	1	3
<u>getCustomerLast()</u>	10	0	1	3

Figure 6-20 EJB method summary with bad response

The EJB Method Summary screen shows how much time it took for each method in the EJB to execute. We see that the method `findCustomerByLastName` took nearly seven minutes, and the method `findByPrimaryKey` took five minutes.

- To see the details of the execution of `findCustomerByLastName`, click the **findCustomerByLastName** link.

The EJB Method Detail page appears, which is shown in Figure 6-21 on page 259.

SNAPSHOT INFO			
APPLICATION SERVER NAME:	WTSCPLX1.WSES.RVA		
RESOURCE:	ERWWW:ERWWWBMPs.jar::CustomerEntity.findCustomerByLastName (java.lang.String,short,short,boolean)		
INTERVAL START/END:	Sun Nov 17 23:58:39 EST 2002 - Mon Nov 18 11:12:17 EST 2002		
EJB METHOD DETAIL			
Method Signature:	findCustomerByLastName (java.lang.String,short,short,boolean)	Invocations:	1
Average Response Time:	409728	Max Response Time:	409728
Transaction Policy:	3	EJB Roles:	
ejbLoad Invocations:	0	ejbLoad Average Execution Time:	0
ejbLoad Maximum Execution Time:	0	ejbStore Invocations:	0
ejbStore Average Execution Time:	0	ejbStore Maximum Execution Time:	0
ejbActivate Invocations:	1	ejbActivate Average Execution Time:	0
ejbActivate Maximum Execution Time:	0	ejbPassivate Invocations:	0
ejbPassivate Average Execution Time:	0	ejbPassivate Maximum Execution Time:	0

Figure 6-21 EJB Method Detail: Shows statistics about the method's invocation history and performance

Further investigation shows us that the method `findCustomerByLastName` is used only by this user, and there are no other invocations of this method in our

database. A closer look at the method reveals that indexing has been turned off for the query, which is the cause of the problem.

6.5.5 Example 7

In Example 7, users are not only experiencing delays, but their transactions are hanging and not returning.

Methodology

Our methodology is as follows:

1. Since we know that there are active transactions in the system, we use In-Flight Request Search to look at all active requests in the Trade2 Application server group. From there, we drill down by getting a stack trace of the thread.
2. We consider taking action to cancel the hanging thread.

Procedure

In this case, since there are active transactions in the system, we use In-Flight Request Search to look at all active requests in the Trade2 Application server group.

To locate hanging requests in the Trade2 Application server group, do the following:

1. Navigate to **Problem Determination -> In-Flight Request Search**.
This brings up the In-Flight Request Search page.
2. On the In-Flight Request Search page, enter the following information in the SEARCH CRITERIA section:
 - Group: Trade2 Application
 - Server: All Servers
 - Search Request: <leave blank>

Click **OK**. The results of the search are shown in Figure 6-22 on page 261.

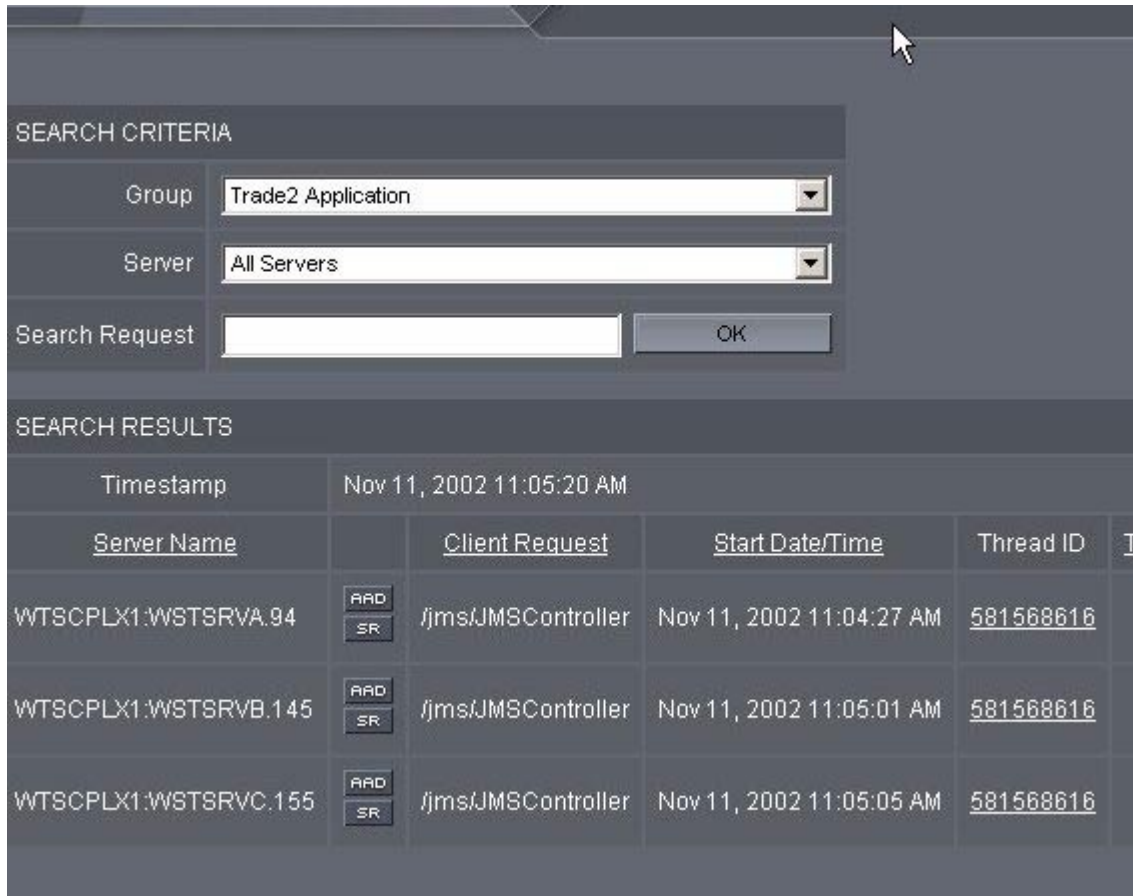


Figure 6-22 In-Flight Request Search: Shows the requests currently executing in the application server

The results of our in-flight request search show several transactions that have been resident in the system for varying amounts of time.

3. To look at a particular request in more detail, click the thread id of the oldest transaction (581568616).

The resulting Request Detail page is shown in Figure 6-23 on page 262.

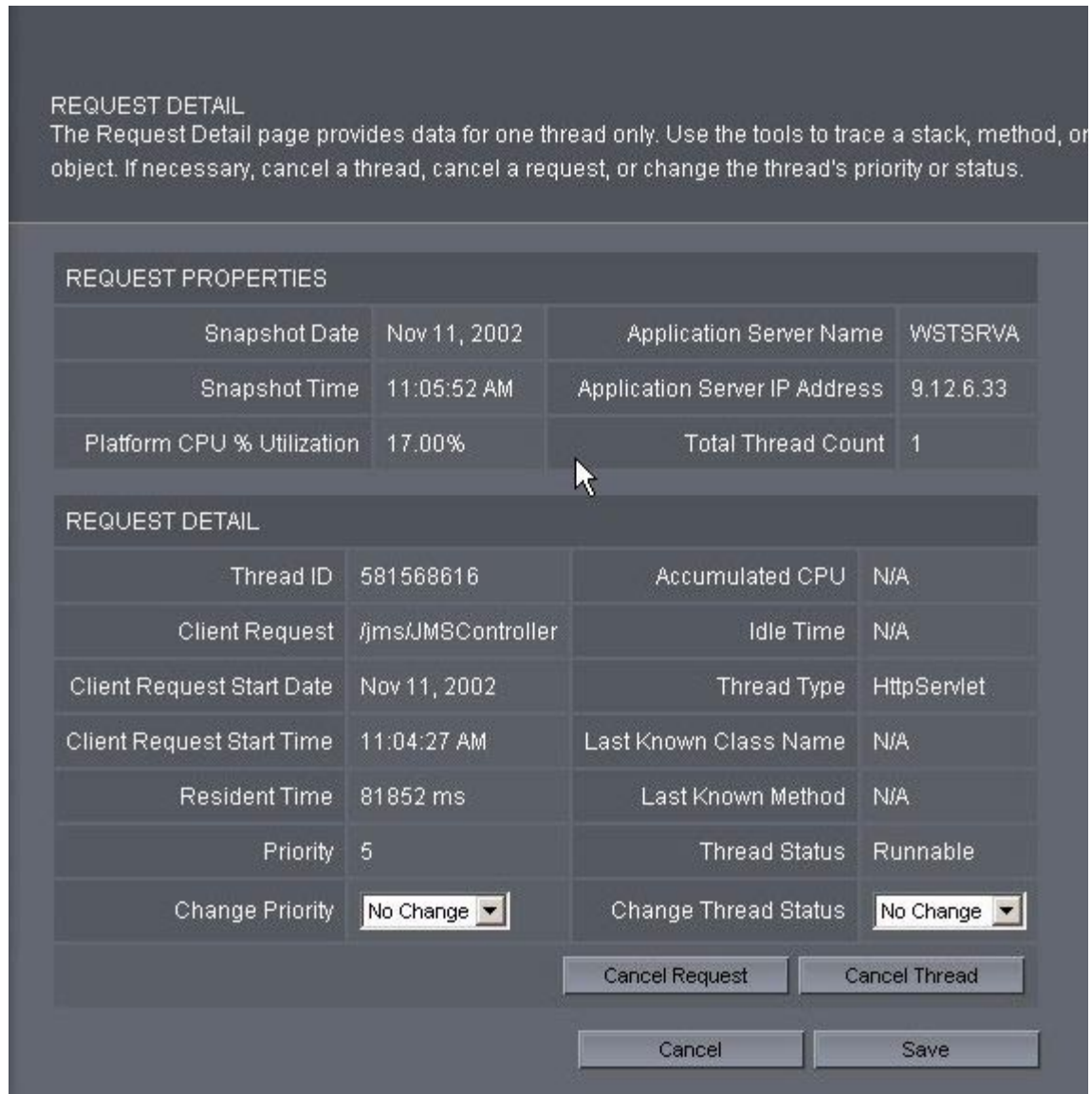


Figure 6-23 Request Detail: Shows detail information on an In-Flight Request

In this Scenario, since we are configured to Monitoring Level 1, there is no method and CPU information available. However, we can view the Stack Trace of the running transaction to see what it is currently doing.

- To view the stack trace of the hanging request, click **Stack Trace** in the left side navigation.

The results of the stack trace are shown in Figure 6-24.

Click **Stack Trace** in the MENU on the left side of the page to view the stack trace of the hanging request. The results of the stack trace are shown in Figure 6-24.

...e lists the JVM stack depth for the methods that have not completed execution. The trace provides the Class
...ames for each level of the stack.

PROPERTIES			
Snapshot Date	Nov 11, 2002	Application Server Name	WSTSRVA
Snapshot Time	11:06:11 AM	Application Server IP Address	9.12.6.33
Form CPU % Utilization	18.00%	Total Thread Count	1

com.ibm.mq.server.MQSESSION	Method	_MQGET
com.ibm.mq.server.MQSESSION	Method	MQGET
com.ibm.mq.MQQueue	Method	get
com.ibm.mq.jms.MQQueueReceiver	Method	getMessage
com.ibm.mq.jms.MQQueueReceiver	Method	receiveInternal
com.ibm.mq.jms.MQQueueReceiver	Method	receive
mqstuff.Client	Method	web
_JMSResults_jsp_0	Method	_jspService
org.apache.jasper.runtime.HttpJspBase	Method	service

Figure 6-24 Stack Trace: Shows the stack of a thread executing a Web request

The stack trace shows that the request is executing an `com.ibm.mq.server.MQSESSION_MQget`. Viewing the other hanging transactions

shows the same result, which means it is very likely that there is something unexpected state with one of the queues.

Check the queues to determine the root cause of the problem.

Follow-up

IBM WebSphere Studio Application Monitor gives you the ability to submit a request to cancel in-flight transactions from the Request Detail screen. This is a feature that should be used only as a last resort, since cancelling threads may leave the JVM in an inconsistent state.

6.5.6 Example 8

In Example 8, an administrator wants to understand the work his servers are doing, since there is a significant increase in the volume of requests.

Methodology

Our methodology is to use Performance Analysis and Reporting to create a Request Analysis report, and then to drill down to see the nature of the workload over the entire period under consideration.

Procedure

In this case we create a Request Analysis report that shows workload decomposition on a minute-by-minute basis.

To determine the composition of requests served by the Trade2 Application server group, do the following:

1. Navigate to **Performance Management -> Performance Analysis & Reporting**.

This brings up the Performance Analysis and Reporting page.

2. Click **Define Report** in the left navigation.

This brings up the Server and Report Type Selection page.

3. Enter the following information in the Server and Report Type Selection section:

- Group: Trade2 Application
- Server: All Servers
- Report Type: Request Analysis

Click **Next>**. This brings up the Report Filtering Options page.

4. Enter the following information in the Report Filtering Options section:

- Metric: Throughput per Minute

- Request Type: all
- Request Name: <leave blank>

Click **Next>**. This brings up the Data Set Parameters page.

5. Enter the following information in the indicated sections:

- Start Date: 9/8/02 12:00 PM
- End Date: 9/8/02 12:00 AM
- Contrast Options: None
- Data Grouping: Minute of the Hour

Click **Finish**. This brings up the Trend Report, which is shown in Figure 6-25 on page 266.

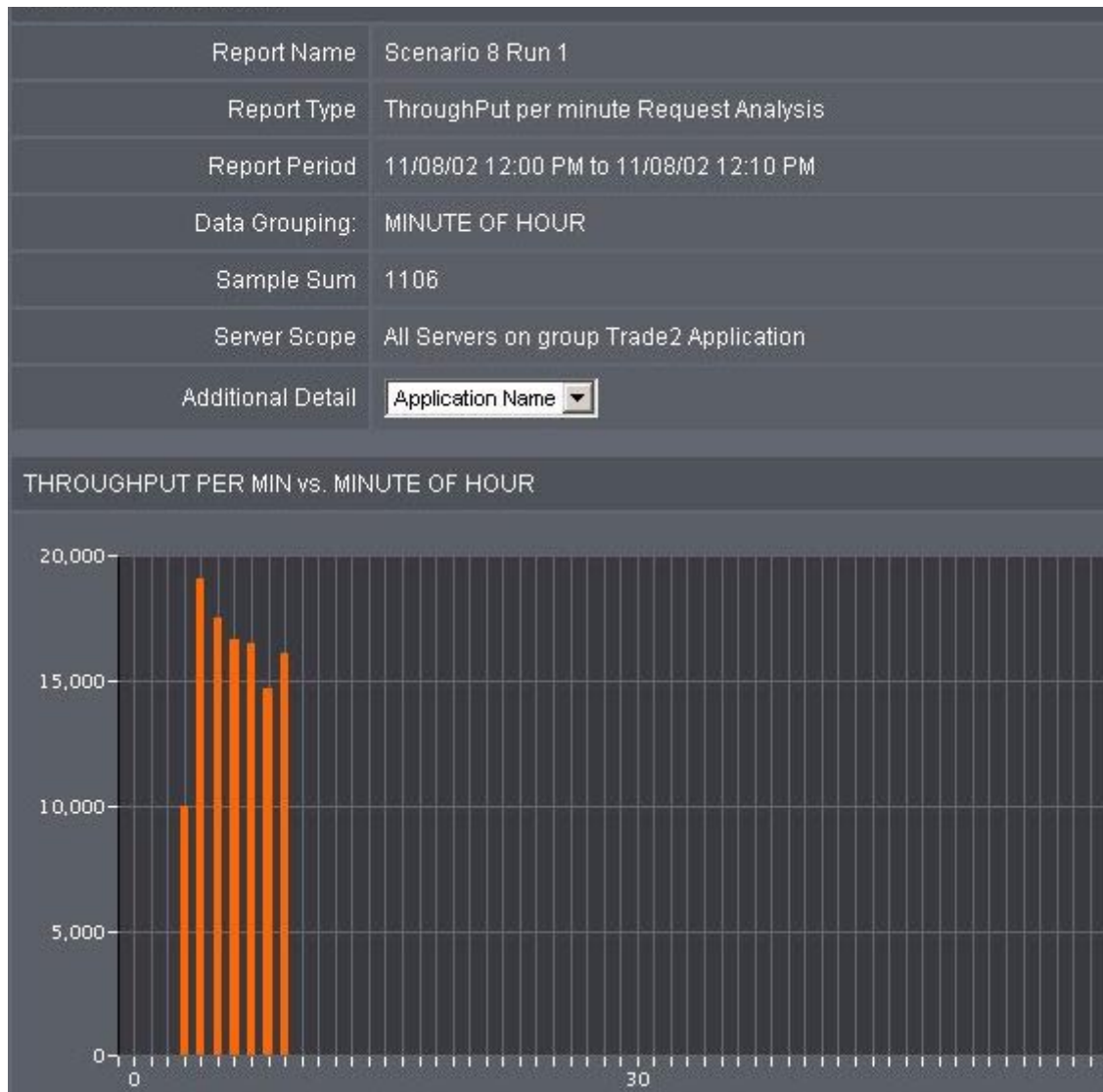


Figure 6-25 Trend Report: Shows the throughput trend for the specified period of time

The Trend Report shows heavy activity for several minutes, starting at minute 4. To see the composition of the requests present during this period of time, do the following:

1. In the Additional Detail field of the Report Properties section, choose **Application Name**.
2. Click the bar in minute 4.

The resulting Decomposition Report is shown in Figure 6-26.

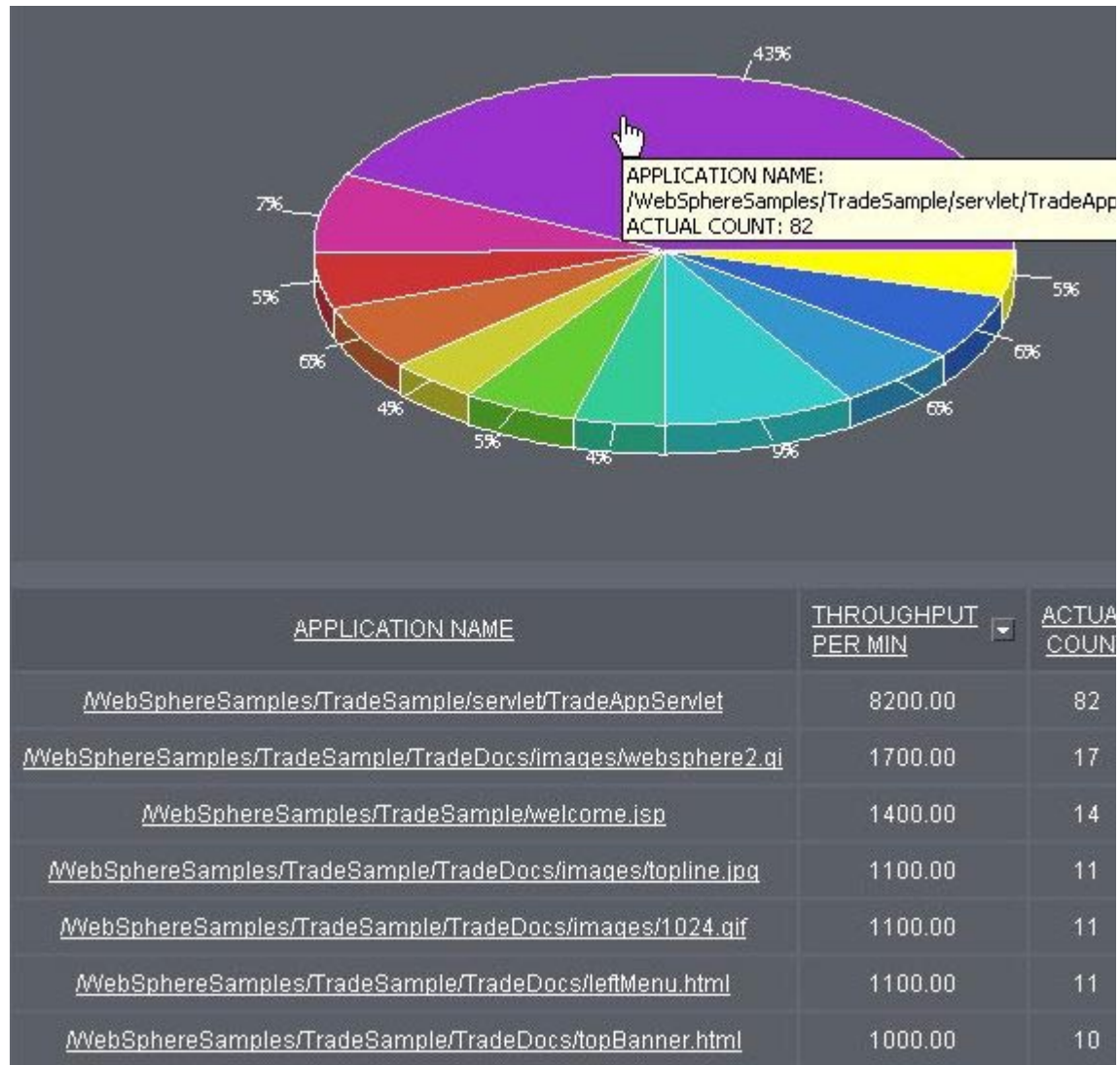


Figure 6-26 Decomposition Report: Shows the requests that were executed during the selected period

The Decomposition Report shows that 43% of the workload are requests for the TradeAppServlet, and approximately 30% of the other requests are for static content.

It is up to back-office performance analysts to determine whether this workload composition is expected. Some enterprises require static files to be served outside of WebSphere, while others do not mind.

6.5.7 Example 9

In Example 9, we see an increase in volume, but no problems.

Methodology

To find volume increases in a broad sense, we use the Application Overview, which gives an overall view of the server farm. We drill down to view individual servers' performance metrics.

Procedure

To find volume increases in the broader level, we use the Application Overview, which gives an overall view of the server farm, and then look at Server Availability Detail for the server group that shows increased activity.

To evaluate the overall health of a server farm, do the following:

1. Navigate to **Availability > Application Overview**.

The Application Overview screen appears, which is shown in Figure 6-27 on page 269. Mousing over the blue bars displays the number of requests and average response time for the interval.

Note: Under normal operation, at the base of the blue bars that represent volume of throughput for the interval, there is an indicator light which indicates how the response time in that interval compares to the baseline. The indicator light uses the colors green, yellow and red. These indicators are absent due to the lack of baseline data.

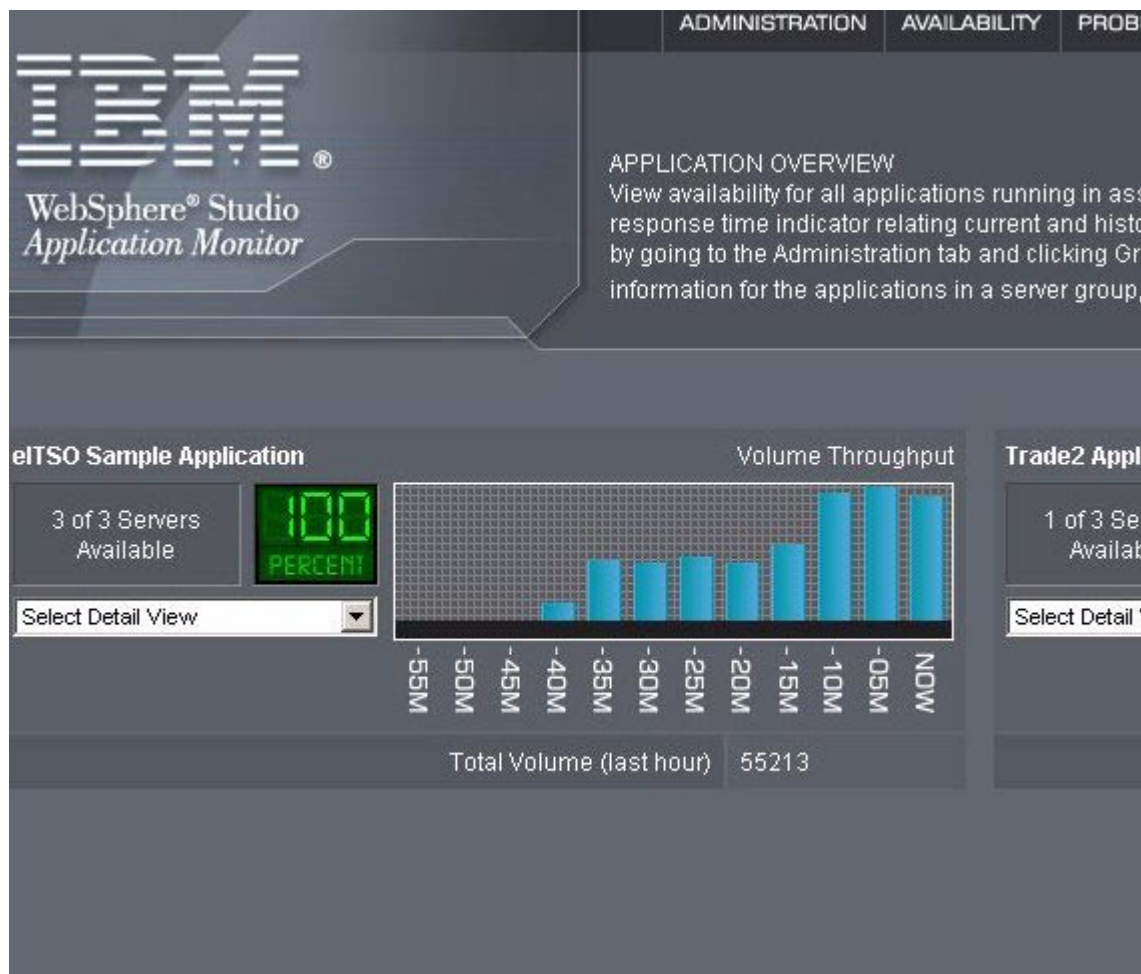


Figure 6-27 Application overview: Volume has been steadily increasing on the eITSO Sample Application servers

The Application Overview shows that the volume of the servers in the eITSO Sample Application group has increased over the last forty minutes.

To monitor the health of the servers in the eITSO Sample Application group, do the following:

2. Select **Server Availability Detail** from the drop-down under the eITSO Sample Application group.

The Server Availability Detail page appears, which is shown in Figure 6-28 on page 270.

REFRESH SETTINGS

Refresh Interval (sec.) Next Refresh (sec.)

SERVER DETAIL

Name		Status	Platform	Volume	Total Volume	Total CPU %	JVM Memory Usage (mb)	Group Name
<input checked="" type="checkbox"/> WTSCPLX1 WSESRVA 110 (L1)	<input type="button" value="ADD"/> <input type="button" value="SR"/> <input type="button" value="SRC"/>	Available	z/OS	0	36	44.00	18	eITSO Sa Applica
<input checked="" type="checkbox"/> WTSCPLX1 WSESRVA 112 (L1)	<input type="button" value="ADD"/> <input type="button" value="SR"/> <input type="button" value="SRC"/>	Available	z/OS	159	26,125	49.00	113	eITSO Sa Applica
<input checked="" type="checkbox"/> WTSCPLX1 WSESRVB 29 (L1)	<input type="button" value="ADD"/> <input type="button" value="SR"/> <input type="button" value="SRC"/>	Available	z/OS	135	22,787	4.00	48	eITSO Sa Applica
<input checked="" type="checkbox"/> WTSCPLX1 WSESRVB 87 (L1)	<input type="button" value="ADD"/> <input type="button" value="SR"/> <input type="button" value="SRC"/>	Available	z/OS	0	25	50.00	18	eITSO Sa Applica
<input checked="" type="checkbox"/> WTSCPLX1 WSESRVC 103 (L1)	<input type="button" value="ADD"/> <input type="button" value="SR"/> <input type="button" value="SRC"/>	Available	z/OS	1	35	14.00	20	eITSO Sa Applica
<input checked="" type="checkbox"/> WTSCPLX1 WSESRVC 28 (L1)	<input type="button" value="ADD"/> <input type="button" value="SR"/> <input type="button" value="SRC"/>	Available	z/OS	183	24,780	70.00	145	eITSO Sa Applica
<input checked="" type="checkbox"/> Clear All								

Figure 6-28 Server Availability Detail: Shows server health, which looks okay.

From the Server Availability Detail screen, we do not see any obvious issues. On the one hand, we can assume that this behavior is good, in that it simply represents an increase in the site's popularity.

On the one hand, we can assume that this behavior is good, in that it simply represents an increase in the site's popularity.

On the other hand, we can repeat what we did in 6.5.6, “Example 8” on page 264 and examine the workload composition both before and after the period of the increase in activity. A large discrepancy between these two workload decompositions might indicate that there is something wrong. If the workload decompositions are similar, then there is likely nothing wrong. In addition, reports can be created to compare the CPU and response time of the requests that were captured in the two time periods.

6.5.8 Example 10

In Example 10, we see cases where the response time for a transaction, which often completes quickly, is slow.

Methodology

Our methodology for problem determination is as follows:

1. Since this is not an in-flight transaction, we start with the Performance Analysis and Reporting section of WSAM to isolate time periods in which requests took a long time to execute.
2. We drill down on the offending time intervals by looking at the requests executed within them, and then drill down further to the underlying method trace. This provides details we can pass back to the developers for further analysis.

This has already been exercised in 6.5.1, “Example 1” on page 237. Instead of presenting the same methodology, we present our methodology for performance management.

Our methodology for performance management is as follows:

1. Wearing the hat of a performance analyst, we want to understand the method distribution of the application, to see if there are methods that end up being called too frequently. We begin by running a Top Methods Used report.
2. Once we identify methods that look suspicious, we run a Method Analysis report and filter for the suspect method. This tells us which requests call the method in question.
3. We proceed to run a Request Analysis report, then drill down on the appropriate request and get a method trace, which we use to help analyze the problem.

Procedure

Wearing the hat of a performance analyst, we want to understand the method distribution of the application, to see if there are methods that end up being called too frequently. We begin by running a Top Methods Used report.

To see if any methods are called too frequently, do the following:

1. Navigate to **Performance Management -> Performance Analysis & Reporting**.

This brings up the Performance Analysis and Reporting page.

2. Click **Define Report** in the left navigation.

This brings up the Server and Report Type Selection page.

3. Enter the following information in the Server and Report Type Selection section:

- Group: eITS0 Sample Application
- Server: All Servers
- Report Type: Top Report

Click **Next>**. This brings up the Report Filtering Options page.

4. Enter the following information in the Report Filtering Options section:

- Metric: Throughput per Minute
- Request Type: all
- Request Name: <leave blank>

Click **Next>**. This brings up the Report and Date Range Selection page.

5. Enter the following information in the Select Type and Date Range section:

- Top Report: Top Methods Used
- Start Date: 11/13/02 8:00 PM
- End Date: 11/13/02 8:30 AM

Click **Finish**. This brings up the Top Methods Used report, which is shown in Figure 6-29 on page 273.

TOP METHODS USED

The Top Methods Used report displays the top unique methods used during the report period and how was used. The report displays up to 100 records.

REPORT PROPERTIES

Report Name	Scenario 10 Run 2 Report A
Report Type	Top Methods Used Analysis
Report Period	11/13/02 8:00 PM to 11/13/02 8:30 PM
Server Scope	All Servers on group eITSO Sample Application

RANK ▲	METHOD NAME	COUNT
1	COM/ibm/db2os390/sqlj/jdbc/DB2SQLJPreparedStatement.executeQuery	825
2	COM/ibm/db2os390/sqlj/jdbc/DB2SQLJPreparedStatement.executeUpdate	408
3	stockEntityPackage/StockEntityKey.equals	275
4	itemEntityPackage/ItemEntityKey.equals	234
5	itemEntityPackage/ItemEntityBean.debugOut	198
6	stockEntityPackage/StockEntityBean.debugOut	180
7	irwwbase/IRWWWBase.debugOut	156
8	warehouseEntityPackage/WarehouseEntityKey.equals	147
9	irwwbase/IRWWWBase.getCurrentTime	138
10	districtEntityPackage/DistrictEntityKey.equals	131

Next >

Figure 6-29 Top Methods Used: Shows the most popular method executions for the selected period

From the Top Methods Used report, we see the distribution of the methods called by the user's application. The highest ranking methods are calls to DB2 and calls related to entity beans, which is expected.

However, there are plenty of calls to a debugOut method. Since it seems suspicious that debugOut is called frequently enough to appear in the top 10 methods called, we investigate further by performing a Method Analysis report.

To perform a Method Analysis report on throughput per minute on the selected time frame, do the following:

1. Navigate to **Performance Management -> Performance Analysis & Reporting**.

This brings up the Performance Analysis and Reporting page.

2. Click **Define Report** in the left navigation.

This brings up the Server and Report Type Selection page.

3. Enter the following information in the Server and Report Type Selection section:

- Group: eITS0 Sample Application
- Server: All Servers
- Report Type: Method Analysis

Click **Next>**. This brings up the Report Filtering Options page.

4. Enter the following information in the Report Filtering Options section:

- Metric: Throughput per Minute
- Method: debugOut
- Request Type: all
- Request Name: <leave blank>

Click **Next>**. This brings up the Data Set Parameters page.

5. Enter the following information among the several sections:

- Start Date: 9/13/02 8:00 PM
- End Date: 9/13/02 8:30 PM
- Contrast Options: None
- Data Grouping: Minute of the Hour

Click **Finish**. This brings up the Trend Report.

To see the composition of the requests present during any particular period of time, do the following:

1. In the Additional Detail field of the Report Properties section, choose **Application Name**.
2. Click on any of the bars on the bar chart in the Trend Report.

This brings up the resulting Decomposition Report, which is shown in Figure 6-30 on page 275.

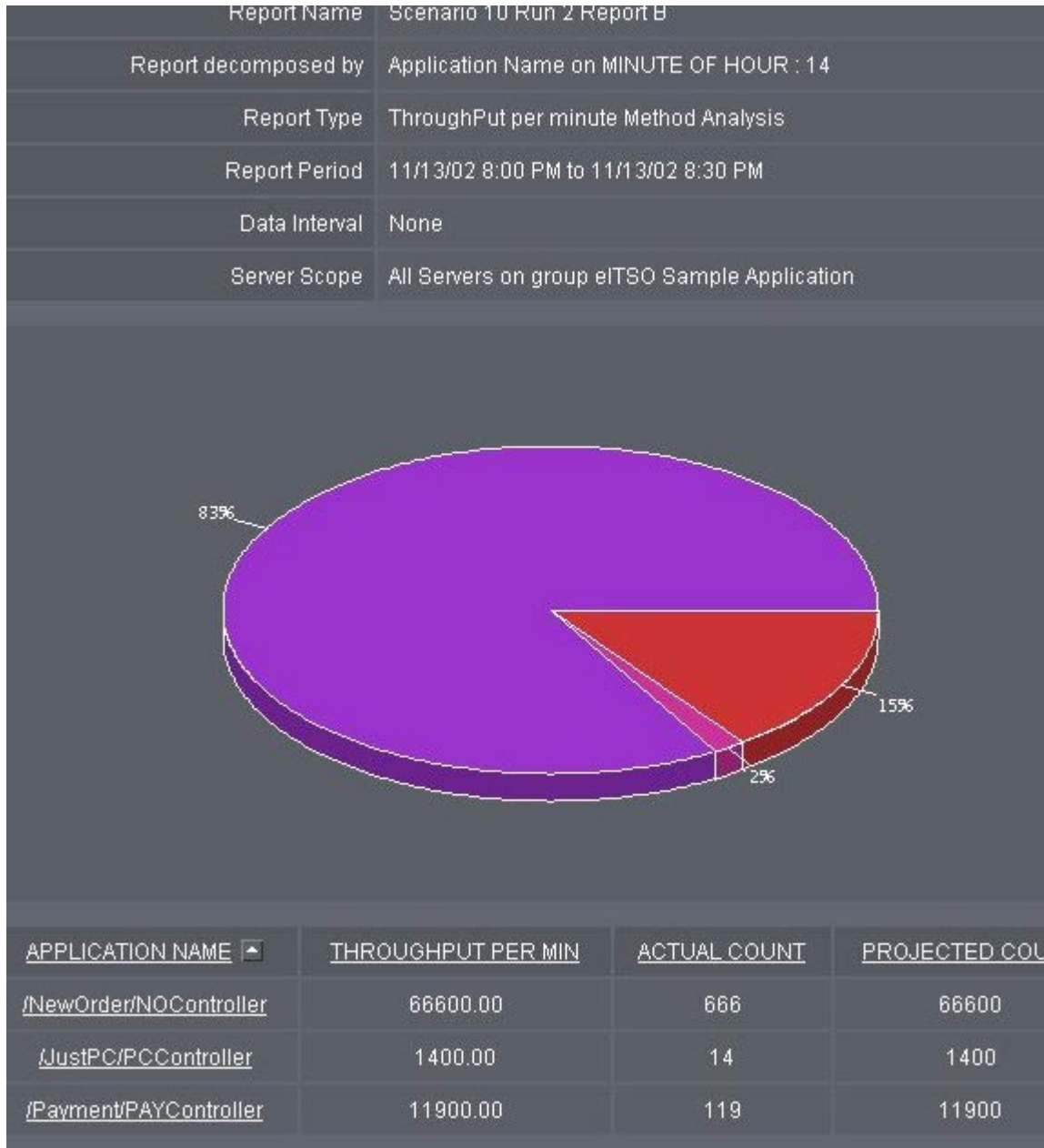


Figure 6-30 Decomposition Report: Gives you the distribution of client requests for the time period

The Decomposition Report shows that most requests containing the debugOut call are handled by the /NewOrder/NOController URI.

Therefore, we want to view the method trace of the URI. To do this, we create a Request Analysis report on throughput per minute of a selected time frame.

To view the method trace of a request on a minute-by-minute basis, do the following:

1. Navigate to **Performance Management -> Performance Analysis & Reporting**.

This brings up the Performance Analysis and Reporting page.

2. Click **Define Report** in the left navigation.

This brings up the Server and Report Type Selection page.

3. Enter the following information in the Server and Report Type Selection section:

- Group: eITSO Sample Application
- Server: All Servers
- Report Type: Request Analysis

Click **Next>**. This brings up the Report Filtering Options page.

4. Enter the following information in the Report Filtering Options section:

- Metric: Throughput per Minute
- Request Type: all
- Request Name: <leave blank>

Click **Next>**. This brings up the Data Set Parameters page.

5. Enter the following information in the indicated sections:

- Start Date: 9/13/02 8:00 PM
- End Date: 9/13/02 8:30 PM
- Contrast Options: None
- Data Grouping: Minute of the Hour

Click **Finish**. This brings up the Trend Report.

From here, we can drill down to see the composition of requests for any chosen time period, as follows:

1. In the Additional Detail field of the Report Properties section, choose **Request Type**.

2. Click on any one of the bars.

This brings up the View Request Detail page.

3. On the View Request Detail page, click any one of the links.

This brings up the View Method Trace, which is shown in Figure 6-31.

Entry	neworderSessionPackage/NewOrderSessionBean.getHome	Date/Time	Nov 13, 2002 8:14:19 PM
Entry	irwwwbase/IRWWWBase.debugOut	Date/Time	Nov 13, 2002 8:14:19 PM
Exit	irwwwbase/IRWWWBase.debugOut	Date/Time	Nov 13, 2002 8:14:19 PM
Entry	irwwwbase/IRWWWBase.debugOut	Date/Time	Nov 13, 2002 8:14:19 PM
Exit	irwwwbase/IRWWWBase.debugOut	Date/Time	Nov 13, 2002 8:14:19 PM
Entry	irwwwbase/IRWWWBase.debugOut	Date/Time	Nov 13, 2002 8:14:19 PM
Exit	irwwwbase/IRWWWBase.debugOut	Date/Time	Nov 13, 2002 8:14:19 PM
Entry	irwwwbase/IRWWWBase.debugOut	Date/Time	Nov 13, 2002 8:14:19 PM
Exit	irwwwbase/IRWWWBase.debugOut	Date/Time	Nov 13, 2002 8:14:19 PM
Entry	irwwwbase/IRWWWBase.debugOut	Date/Time	Nov 13, 2002 8:14:19 PM
Exit	irwwwbase/IRWWWBase.debugOut	Date/Time	Nov 13, 2002 8:14:19 PM
Entry	irwwwbase/IRWWWBase.debugOut	Date/Time	Nov 13, 2002 8:14:19 PM
Exit	irwwwbase/IRWWWBase.debugOut	Date/Time	Nov 13, 2002 8:14:19 PM
Entry	irwwwbase/IRWWWBase.debugOut	Date/Time	Nov 13, 2002 8:14:19 PM
Exit	irwwwbase/IRWWWBase.debugOut	Date/Time	Nov 13, 2002 8:14:19 PM
Entry	irwwwbase/IRWWWBase.debugOut	Date/Time	Nov 13, 2002 8:14:19 PM
Exit	irwwwbase/IRWWWBase.debugOut	Date/Time	Nov 13, 2002 8:14:19 PM
Entry	irwwwbase/IRWWWBase.debugOut	Date/Time	Nov 13, 2002 8:14:19 PM
Exit	irwwwbase/IRWWWBase.debugOut	Date/Time	Nov 13, 2002 8:14:19 PM
Entry	irwwwbase/IRWWWBase.debugOut	Date/Time	Nov 13, 2002 8:14:19 PM
Exit	irwwwbase/IRWWWBase.debugOut	Date/Time	Nov 13, 2002 8:14:19 PM
Entry	irwwwbase/IRWWWBase.debugOut	Date/Time	Nov 13, 2002 8:14:19 PM
Exit	irwwwbase/IRWWWBase.debugOut	Date/Time	Nov 13, 2002 8:14:19 PM

Figure 6-31 Method Trace: Shows numerous calls to the debugOut method, probably more than necessary

The View Method Trace report shows that the method debugOut is called a lot, and is a likely candidate for causing performance problems if the application is not configured correctly.



A

Java and J2EE details

This appendix provides additional details on the way Java and J2EE applications behave. Primarily for the benefit of those not familiar with the Java world, it expands on the concepts introduced in “The WebSphere programming model” on page 7.

A.1 Java class loading

Java loads execution code in a lazy manner. As a Java application executes, it encounters references to classes and methods (because, fundamentally, all code exists within methods and methods are grouped into classes; all other constructs are primarily for packaging purposes). If the class containing the referenced method is already available, that code is executed. However, if the referenced class is not available, Java will use a *class loader* to locate the referenced class into memory and make it available. Java will then execute the referenced method. This algorithm ensures that as long as a class is available, it will only be loaded once.

WebSphere has multiple class loaders to load various pieces of itself, extensions, and application code. Each of these class loaders has a search policy, known as a *class path*, to locate classes. A class path is a list of JAR files to be searched in some order until the referenced class is found. Once a class loader finds the referenced class, it will load it. The class loader makes no attempt to discern multiple, identically named classes on the class path. Although there is no performance impact to this search behavior, it can introduce subtle bugs in the function of an application if multiple copies of the identically named classes are on a class path.

A.2 Java runtime execution

As mentioned in 1.2.1, “Java overview” on page 7, Java classes are not stored as object code in the sense that the hardware understands it. They are stored as byte codes to be interpreted by the JVM.

In order to achieve good runtime performance, the JVM employs an in-memory compiler that converts Java byte-code into object code at application run time. This technique is known as just-in-time (JIT) compilation, and can have performance benefits over traditional, static compilation. JIT compilers are capable of analyzing the runtime execution of a set of code, and they employ optimizations based on information unavailable to static compilers.

One common optimization is called *method inlining*. As an example of this, the JIT compiler notices that method A invoked method B often; instead of creating object code to perform a subroutine call, the JIT compiler produces object code that includes method B in the same execution path as method A.

The JVM has multiple runtime options. The JIT compiler can be disabled, but this is normally not done in production environments because of the dramatic decrease in performance. The JVM also has a runtime debugging mode known as Java Platform Debugger Architecture (JPDA). This allows a debugger to

remotely connect to a running JVM and perform development debugging tasks. This is rarely used in production, although it can be helpful. Be aware that using JPDA can force the JVM to disable the JIT compiler.

Another runtime option is the Java Virtual Machine Profiler Interface (JVMPPI). JVMPPI is intended for development-time profiling of an application, although it is sometimes used in production to gather runtime performance metrics. JVMPPI does not disable the JIT compiler, although the use of certain JVMPPI features (such as method execution timing and lock contention monitoring) can force the JIT compiler not to optimize certain code paths that it would otherwise. In particular, using JVMPPI can prevent the JIT compiler from inlining some methods. The performance impact of JVMPPI can vary widely between versions of JVMs and Java applications.

A.3 Java memory and garbage collection

Like many other languages, Java provides a facility for allocating memory on a heap. In fact, unlike some other languages, the Java programming model encourages most memory allocations to be on the heap. In order to make use of a Java class (and, consequently, the class's methods), that class must first be instantiated. An instantiated class is known as an *object*. In Java, all object instantiations are done through heap allocations. Unlike C++, Java objects cannot be instantiated on the stack. In general, stack memory allocation is much faster than heap allocation, but modern JVMs, particularly the IBM JVMs, recognize Java's inherent need for fast heap allocation and heavily optimize the internal layouts of the heap and memory allocation structures to achieve very good performance.

To facilitate some of its heap allocation optimizations, the JVM will request a large block of memory from z/OS at startup. After this initial request, the JVM does not request additional storage from z/OS for heap memory. The JVM allocates memory from this block to the application as required. The initial amount of memory that the JVM requests from z/OS is called the *maximum heap size* and set by JVM_HEAPSIZE in the Current.env file.

The normal life cycle of a memory block from the Java heap is:

1. Allocate a block of memory and create an explicit Java reference to that block.
2. Using the explicit Java reference, perform operations on that block of memory.
3. Remove all references to that memory.
4. De-allocate the unreferenced block of memory and return it to the available heap.

The first three stages of the memory block life cycle are explicit within the Java application code. At step three, the allocated memory block is referred to as “garbage” because, without any references to the memory block, the application can no longer use that memory block. Step four occurs automatically through a JVM internal component called the “garbage collector”. At periodic points, the garbage collector will collect and return the memory blocks to the pool of available heap memory. Also, during the course of using and garbage collecting memory, the heap can become fragmented. The garbage collector will occasionally compact the heap in order to provide better performance for future memory allocations.

There are algorithms in the garbage collector which will attempt to only use the portion of the JVM heap that is really needed.

If the minimum heap size is set to something less than the maximum heap size, the JVM will attempt to only use the minimum heap size until too much of this subset of the heap is filled with live objects. Additional segments of the Java heap will be used as needed until the maximum heap size is reached. The memory requirements may look like Figure A-1.

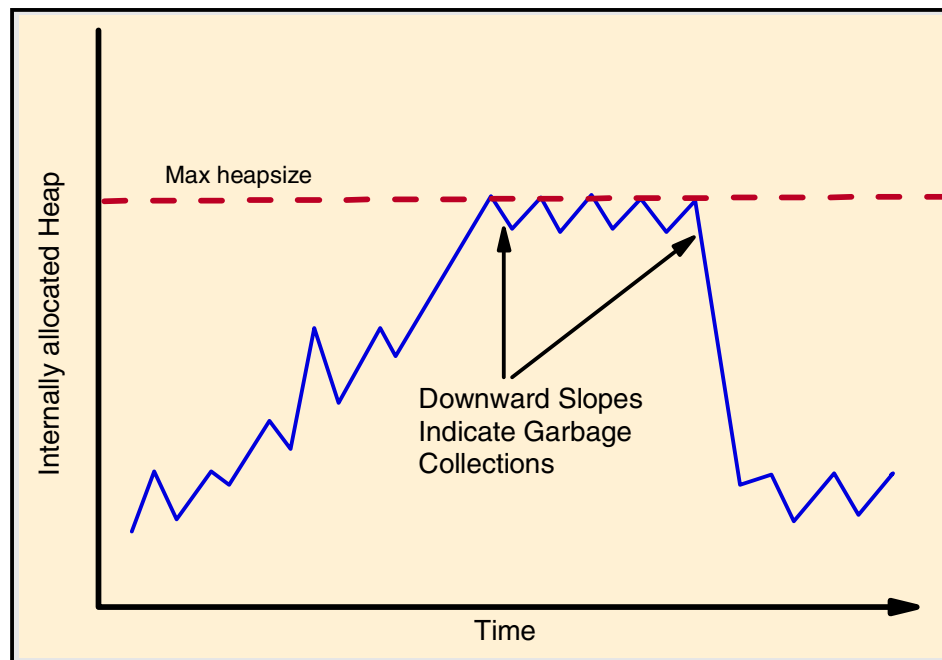


Figure A-1 JVM allocation with Min less than Max heapsize

However, on z/OS we have found that the best performance is generally achieved by setting the minimum heap size to the same value as the maximum heap size. This reduces the overhead of garbage collection in general, and it ensures that the full heap is exploited, often allowing more time between garbage collections and thereby reducing GC overhead. The memory requirements may look like Figure A-2.

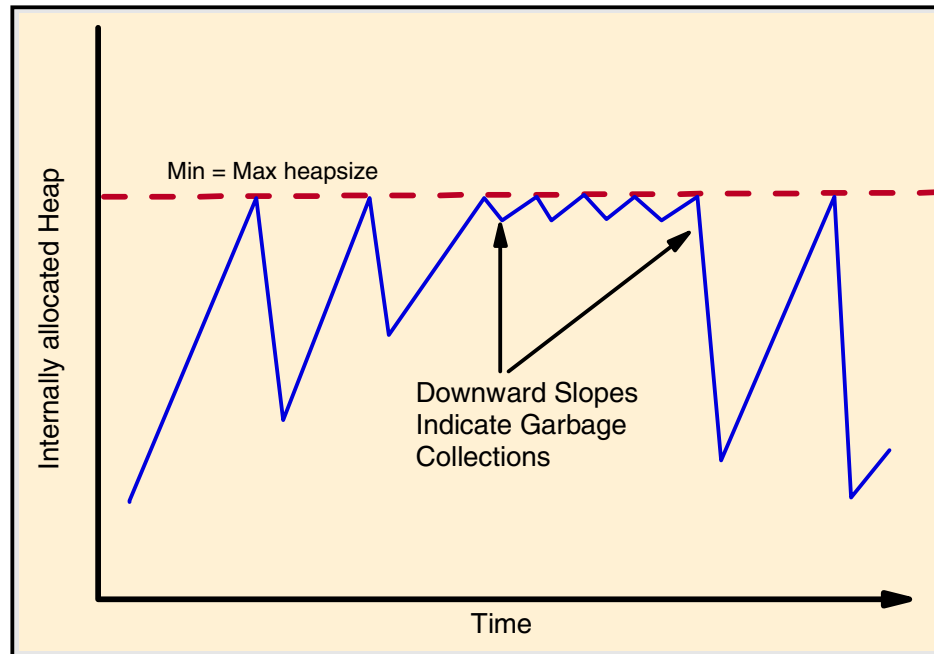


Figure A-2 JVM allocation with Min equal Max heap size

In the WebSphere 4.01 environment on z/OS, Workload Manager will start and stop server regions as needed to meet client demand. Because Workload Manager is aware of the availability of real storage and other system resources, this z/OS system component is able to factor system resource information into the decision whether or not to start an additional server region. This is a much better way to manage real storage consumption than trying to get the JVM to only use the heap storage it really needs.

Activity of the garbage collector can be monitored using the verbose GC option. Running native Java on z/OS, verbose GC data collection can be requested by using the option `-verbose:gc`. Under WebSphere 4.01 for z/OS, this option can be enabled by adding the following option to your `current.env` file.

```
JVM_ENABLE_VERBOSE_GC=1
```

For more information about garbage collection on z/OS, go to the Web site for Java on z/OS:

<http://www-1.ibm.com/servers/eserver/zseries/software/java/>

Look for Garbage Collection under Hints and Tips.

For more information on Java memory performance and the garbage collector in general, see:

<http://www-106.ibm.com/developerworks/library/i-garbage1>

<http://www-106.ibm.com/developerworks/library/j-berry.index.html>

<http://www-1.ibm.com/support/techdocs/atmastr.nsf/WebIndex/WP100292>

http://www-1.ibm.com/servers/eserver/zseries/software/java/gcn2_faq.html

<http://www-1.ibm.com/support/techdocs/atmastr.nsf/WebIndex/TD100748>

Memory leaks

If a Java application repeatedly allocates memory that cannot be collected by the garbage collector, eventually the JVM will not be able to free enough heap memory through garbage collection to satisfy a request to instantiate an object in the heap. At this point, the JVM is in an out-of-memory condition and the JVM will terminate. Even with a very large heap, you can get an out of memory condition if your application has a memory leak.

A memory leak occurs when an application does not remove references to objects in the heap when it should. For example, an application might put pointers to objects in a hashtable. If these pointers are not cleaned up correctly when the objects are no longer needed, this could result in a memory leak. Since the garbage collector still sees references to these old objects, they will not be cleaned up, even though the application no longer needs them.

A verbose GC trace will give a clear indication whether or not you have a memory leak. It does this by showing the free space after each GC. If the free space after each GC declines over time, a memory leak is likely.

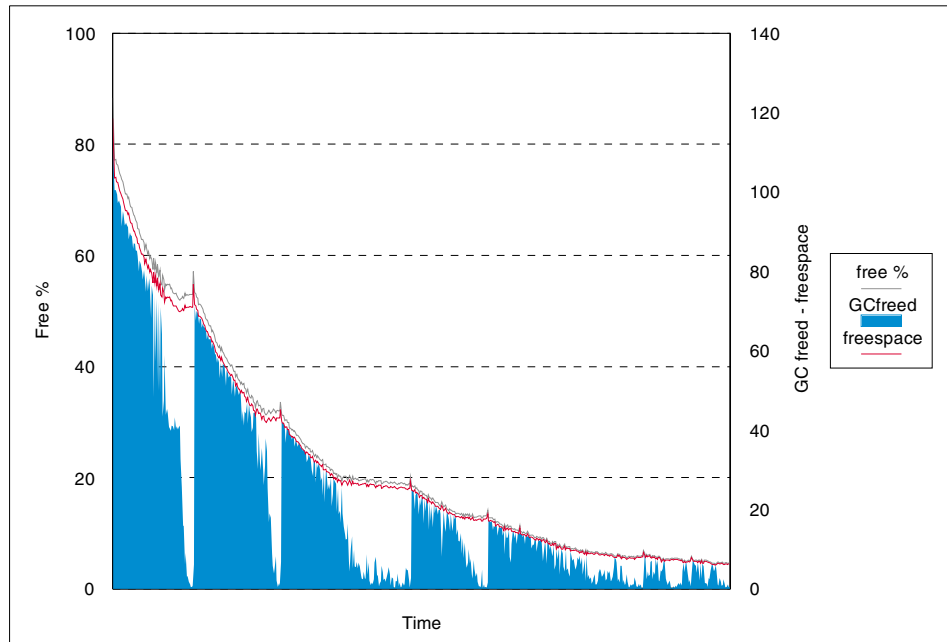


Figure A-3 Decline in free space over time

Managing memory leaks in production

Eventually, an application with a memory leak will crash the entire server region and take down all of the applications hosted with that JVM. At some point before the server region crashes, response times for all of its applications will increase dramatically because the garbage collector is spending more and more time performing its function. The time at which the server region becomes unusable is generally not predictable nor is the time at which it will crash predictable. In general, knowing either of these times requires knowing the load on the system, the available memory in the JVM, the exact nature of the memory leak, and the rate at which the leak occurs.

In production, the only option to manage a memory leak is to recycle the entire server region in a controlled manner to minimize the impact on users. In cases where the server region hosts many different applications, it is very helpful to isolate the memory leak to a particular application and separate that application into its own server region. Then, with WebSphere 4.01 for z/OS, you can set the Server Recycling Interval to the number of transactions which should be processed before a server region is recycled. When the server region is recycled, a new server region will start, new work will go to that new server region, and the old server region will terminate after completing all the transactions that have already started.

A.4 J2EE application flow

According to the J2EE specification, code that exists in one container cannot directly reference code that exists in another container. In most J2EE applications, this means that servlets and JSPs cannot directly invoke the necessary session beans. Further, according to the specification, no one EJB can directly reference another EJB. This means that session beans cannot directly invoke the necessary entity beans. All references between containers and between EJBs must pass through proxies. These proxies allow transparent remote invocation of services, and also allow WebSphere to track transaction contexts.

To obtain a proxy, the application code must request a resource from WebSphere's JNDI server. If the JNDI server is aware of the resource, it sends a proxy for that resource to the requesting code. If it is not aware of the resource, the requesting code gets an exception.

Figure A-4 on page 286 contains a diagram of this particular application flow.

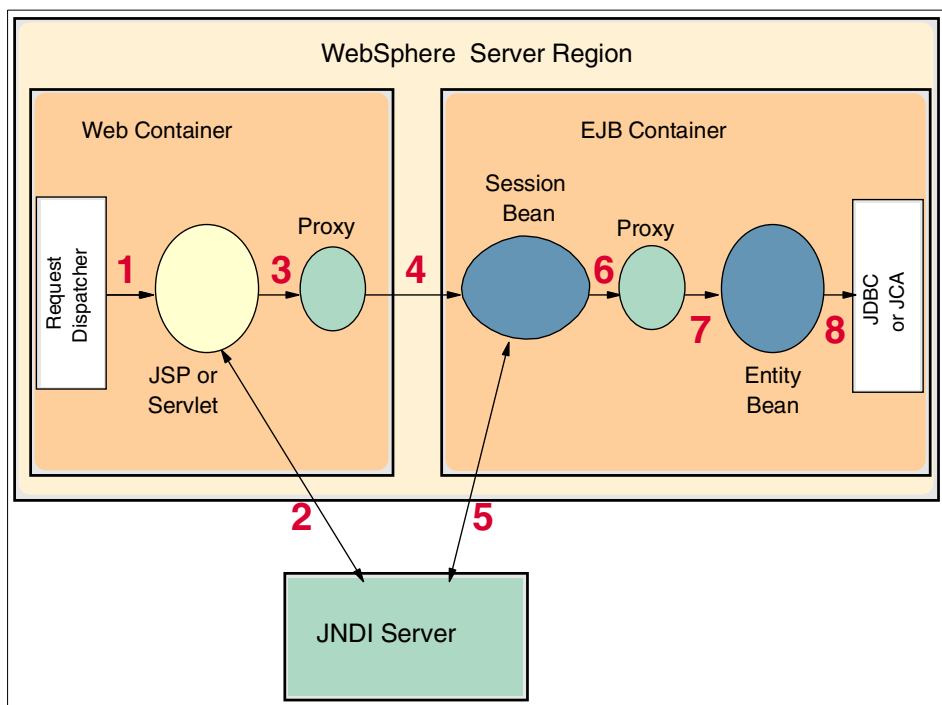


Figure A-4 Use of JNDI and proxies

This is what should happen:

1. The Request Dispatcher sends the HTTP request to a particular servlet or JSP.
2. The servlet or JSP requests a Session Bean from the JNDI Server. The JNDI Server responds with the name of a proxy.
3. The servlet or JSP makes a service request of the Session Bean proxy.
4. The proxy forwards the request to the Session Bean.
5. The Session Bean requests an Entity Bean from the JNDI Server. The JNDI Server responds with the name of a proxy.
6. The Session Bean makes a service request of the Entity Bean proxy.
7. The proxy forwards the request to the Entity Bean.
8. The Entity Bean makes the appropriate request to the appropriate JCA connector for a back-end resource.

One of the primary reasons for JNDI and the use of proxies is to enable distributed transaction management. For example, if the session beans were deployed to a different address space than the servlets and JSPs, the proxy mechanism would automatically forward the request as a Remote Method Invocation (RMI) request by the server hosting the appropriate EJB container.

Most applications are deployed with the Web container that hosts all of the servlets and JSPs in the same address space as the EJB container that hosts all of the session and entity beans. Therefore, WebSphere provides an option, `NoLocalCopies`, to instruct the proxies to send requests directly to the appropriate resource instead of through RMI and (perhaps) the network. The `NoLocalCopies` option allows you to pass objects by reference rather than by value. It helps make local calls faster, and remote calls are still possible when the option is enabled. In many applications that adhere to the J2EE specification, this gives a performance advantage.

Even when the Web container and EJB container are local to the same address space, the application must look up resources using JNDI and use proxies in order to be in compliance with the J2EE specification. While it is possible to use a cache to avoid many JNDI lookups, the use of proxies can introduce undesirable overhead.

Therefore, many applications are not J2EE-compliant and do not use either Session or Entity beans. Instead, they will deploy all code to the Web container. The servlets and JSPs will make direct references to custom code that invokes JCA or JDBC to access back-end systems; see Figure A-5 on page 288.

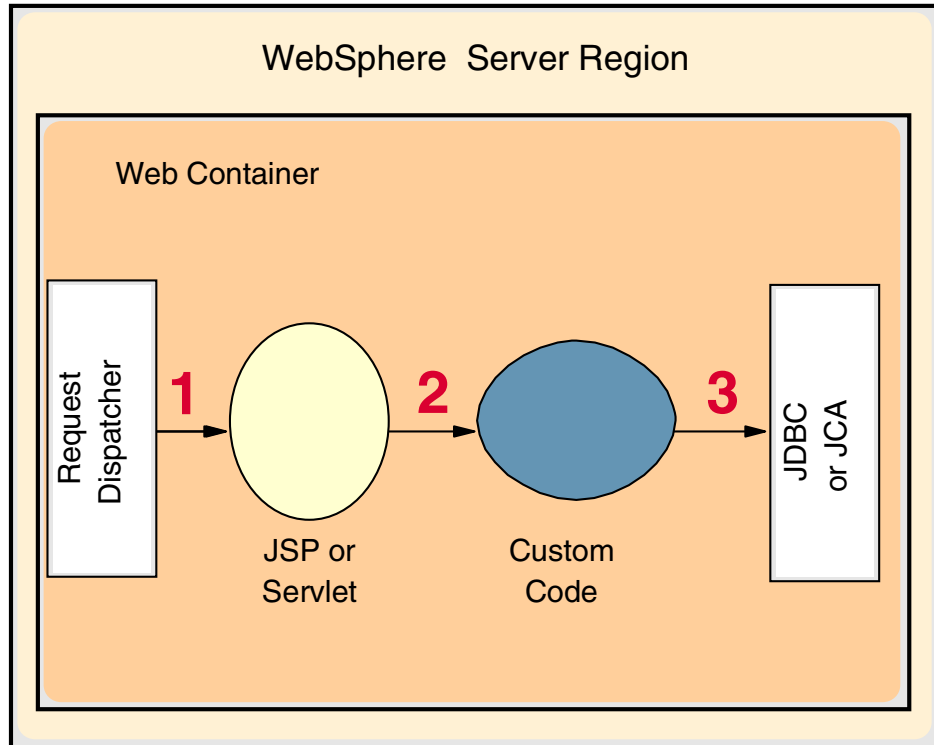


Figure A-5 Use of custom code without JNDI or proxies

A.5 J2EE application structure

An application that follows the J2EE specification will have an internal structure similar to the one shown in Figure A-6 on page 289.

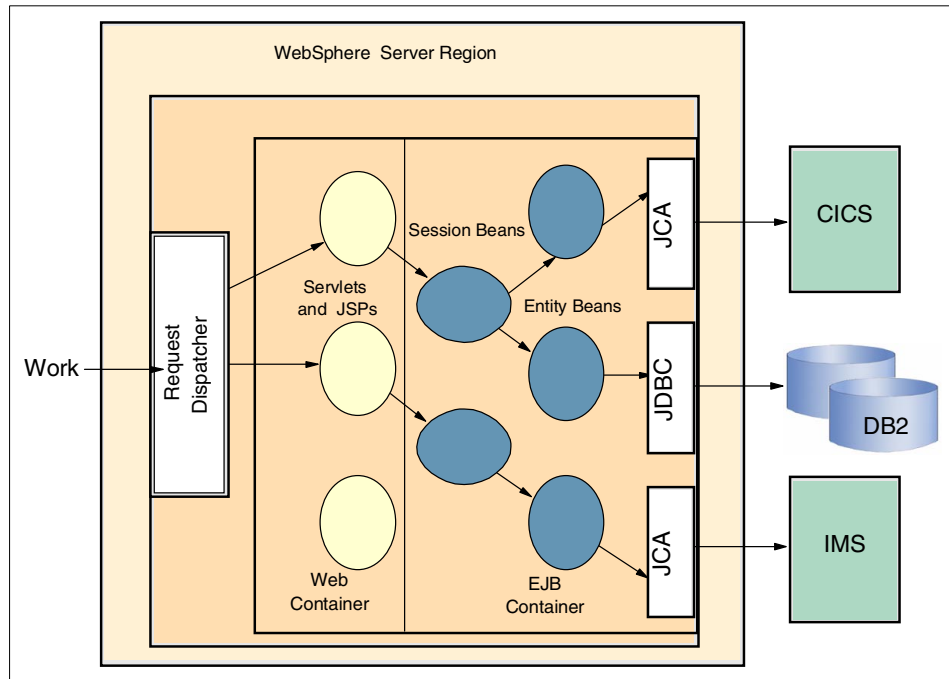


Figure A-6 Quintessential WebSphere application structure

An HTTP request, typically originating from a user's browser, is passed to WebSphere by means (described in "Putting it together: a typical customer installation" on page 16) controlled by Workload Manager (WLM). WebSphere evaluates the HTTP request and dispatches it to the appropriate servlet or JSP. The servlet or JSP then repackages the request in such a way that it can easily be processed by the rest of the application code, and sends it to a session bean for processing. The logic for a particular transaction is contained within a Session Bean. Any back-end data access is handled through entity beans using JCA or JDBC connectors to obtain data from these back-end stores.

Neither WebSphere nor Java enforces any particular internal structure on an application, however. Many applications choose to follow only a portion of the J2EE specification and may use custom code that does not follow any particular specification. In particular, it is common for applications not to use Entity Beans or Session Beans. The internal structure of these applications is described in Figure A-7 on page 290.

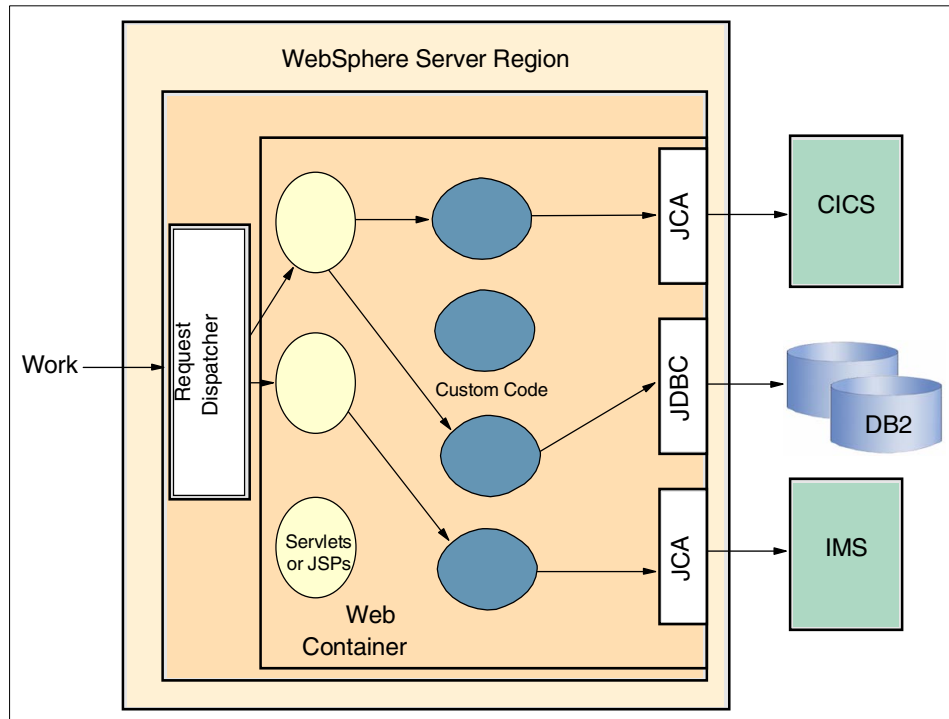


Figure A-7 Custom WebSphere application structure

The HTTP request is still passed to the WebSphere server region based on WLM advice and is dispatched to a server or JSP through the Request Dispatcher. The servlet or JSP then invokes custom code that performs the transaction logic interacting directly with back-end resources via the necessary JCA or JDBC connectors.



B

Configuration files

In this appendix we provide some of the definitions we used in our performance tests. We have included only those definitions most relevant to the topics we describe in the body of the book.

B.1 HTTP Server definitions

Example B-1 shows the principal changes we made to the HTTP server configuration (httpd.conf) on the Windows 2000 server. The plugin was already installed, so we just had to get the HTTP server to listen to all the ports that we were using to distinguish between the eight application servers.

Example: B-1 HTTP server configuration extract

```
Listen 4040
Listen 5050
Listen 6060
Listen 7070
Listen 4050
Listen 5060
Listen 6070
Listen 7080
```

The actual configuration was done using the Administration Server GUI. We selected **Base Settings, Advanced properties**, and then **Specify Additional Ports and IP Addresses**.

The plugin configuration file is shown in Example B-2 in its entirety:

Example: B-2 HTTP server plugin definitions

```
<Config>

    <!-- The LogLevel controls the amount of information that gets written to
         the plugin log file. Possible values are Error, Warn, and Trace. -->

    <Log Name="C:/WebSphere/AppServer/logs/native.log" LogLevel="Error"/>

    <!-- URI groups provide a mechanism of grouping URIs together. Only
         the context root of a web application needs to be specified unless
         you want to restrict the request URIs that get passed to the
application
         server. -->
    <!-- In this game we distribute by virtual host name only, so all URIs are
equal. -->

    <UriGroup Name="default_host_URIs">
        <Uri Name="/WebSphereSamples/*"/>
        <Uri Name="/*"/>
    </UriGroup>

    <!-- Virtual host groups provide a mechanism of grouping virtual hosts
together. -->
```

```
<!-- In this game we have four WAS groups on the z/OS sysplex, hence four
virtual host groups. -->
```

```
<VirtualHostGroup Name="Franck">
  <VirtualHost Name="*:7070"/>
</VirtualHostGroup>
```

```
<VirtualHostGroup Name="FranckTest">
  <VirtualHost Name="*:7080"/>
</VirtualHostGroup>
```

```
<VirtualHostGroup Name="CandleTrade">
  <VirtualHost Name="*:6060"/>
</VirtualHostGroup>
```

```
<VirtualHostGroup Name="WilyTrade">
  <VirtualHost Name="*:5050"/>
</VirtualHostGroup>
```

```
<VirtualHostGroup Name="CyaneaTrade">
  <VirtualHost Name="*:4040"/>
</VirtualHostGroup>
```

```
<VirtualHostGroup Name="CandleeRWW">
  <VirtualHost Name="*:6070"/>
</VirtualHostGroup>
```

```
<VirtualHostGroup Name="WilyeRWW">
  <VirtualHost Name="*:5060"/>
</VirtualHostGroup>
```

```
<VirtualHostGroup Name="CyaneaRWW">
  <VirtualHost Name="*:4050"/>
</VirtualHostGroup>
```

```
<!-- Server groups provide a mechanism of grouping servers together. -->
```

```
<!-- Here we have one Server Group per WAS group. Each group has the same
address but a unique port.
```

```
For each Group, the Cluster address is the distributed VIPA and the
Server addresses are the static VIPAs. -->
```

```
<ServerGroup Name="FMISrvr">
```

```
<ClusterAddress name="haplex1">
  <Transport hostname="202.5.10.11" port="7070" protocol="http"/>
</ClusterAddress>
```

```
<Server CloneID="FMISRV.FMISRVA" Name="SC48">
```

```

        <Transport Hostname="202.5.10.4" Port="7070" Protocol="http"/>
    </Server>

    <Server CloneID="FMISRV.FMISRVB" Name="SC50">
        <Transport Hostname="202.5.10.6" Port="7070" Protocol="http"/>
    </Server>

    <Server CloneID="FMISRV.FMISRVC" Name="SC52">
        <Transport Hostname="202.5.10.7" Port="7070" Protocol="http"/>
    </Server>

</ServerGroup>

<ServerGroup Name="FMXSrvr">

    <ClusterAddress name="haplex1">
        <Transport hostname="202.5.10.12" port="7080" protocol="http"/>
    </ClusterAddress>

        <Server CloneID="FMESRV.FMESRVA" Name="SC48">
            <Transport Hostname="202.5.10.4" Port="7080" Protocol="http"/>
        </Server>

        <Server CloneID="FMESRV.FMESRVB" Name="SC50">
            <Transport Hostname="202.5.10.6" Port="7080" Protocol="http"/>
        </Server>

        <Server CloneID="FMESRV.FMESRVC" Name="SC52">
            <Transport Hostname="202.5.10.7" Port="7080" Protocol="http"/>
        </Server>

</ServerGroup>

<ServerGroup Name="OMESrvr">

    <ClusterAddress name="haplex1">
        <Transport hostname="202.5.10.11" port="6060" protocol="http"/>
    </ClusterAddress>

        <Server CloneID="OMTSRV.OMTSRVA" Name="SC48">
            <Transport Hostname="202.5.10.4" Port="6060" Protocol="http"/>
        </Server>

        <Server CloneID="OMTSRV.OMTSRVB" Name="SC50">
            <Transport Hostname="202.5.10.6" Port="6060" Protocol="http"/>
        </Server>

        <Server CloneID="OMTSRV.OMTSRVC" Name="SC52">
            <Transport Hostname="202.5.10.7" Port="6060" Protocol="http"/>
        </Server>

```



```

        </Server>
    </ServerGroup>
    <ServerGroup Name="INTSrvr">
        <ClusterAddress name="haplex1">
            <Transport hostname="202.5.10.11" port="5050" protocol="http"/>
        </ClusterAddress>
        <Server CloneID="INTSRV.INTSRVA" Name="SC48">
            <Transport Hostname="202.5.10.4" Port="5050" Protocol="http"/>
        </Server>
        <Server CloneID="INTSRV.INTSRVB" Name="SC50">
            <Transport Hostname="202.5.10.6" Port="5050" Protocol="http"/>
        </Server>
        <Server CloneID="INTSRV.INTSRVC" Name="SC52">
            <Transport Hostname="202.5.10.7" Port="5050" Protocol="http"/>
        </Server>
    </ServerGroup>
    <ServerGroup Name="WSMSrvr">
        <ClusterAddress name="haplex1">
            <Transport hostname="202.5.10.11" port="4040" protocol="http"/>
        </ClusterAddress>
        <Server CloneID="WSTSRV.WSTSRVA" Name="SC48">
            <Transport Hostname="202.5.10.4" Port="4040" Protocol="http"/>
        </Server>
        <Server CloneID="WSTSRV.WSTSRVB" Name="SC50">
            <Transport Hostname="202.5.10.6" Port="4040" Protocol="http"/>
        </Server>
        <Server CloneID="WSTSRV.WSTSRVC" Name="SC52">
            <Transport Hostname="202.5.10.7" Port="4040" Protocol="http"/>
        </Server>
    </ServerGroup>
    <ServerGroup Name="OMXSrvr">
        <ClusterAddress name="haplex1">
            <Transport hostname="202.5.10.12" port="6070" protocol="http"/>
        </ClusterAddress>

```

```

    <Server CloneID="OMESRV.OMESRVA" Name="SC48">
      <Transport Hostname="202.5.10.4" Port="6070" Protocol="http"/>
    </Server>

  <Server CloneID="OMESRV.OMESRVB" Name="SC50">
    <Transport Hostname="202.5.10.6" Port="6070" Protocol="http"/>
  </Server>

  <Server CloneID="OMESRV.OMESRVC" Name="SC52">
    <Transport Hostname="202.5.10.7" Port="6070" Protocol="http"/>
  </Server>

</ServerGroup>

<ServerGroup Name="INXSrvr">

  <ClusterAddress name="haplex1">
    <Transport hostname="202.5.10.12" port="5060" protocol="http"/>
  </ClusterAddress>

    <Server CloneID="INESRV.INESRVA" Name="SC48">
      <Transport Hostname="202.5.10.4" Port="5060" Protocol="http"/>
    </Server>

    <Server CloneID="INESRV.INESRVB" Name="SC50">
      <Transport Hostname="202.5.10.6" Port="5060" Protocol="http"/>
    </Server>

    <Server CloneID="INESRV.INESRVC" Name="SC52">
      <Transport Hostname="202.5.10.7" Port="5060" Protocol="http"/>
    </Server>

  </ServerGroup>

<ServerGroup Name="WSXSrvr">

  <ClusterAddress name="haplex1">
    <Transport hostname="202.5.10.12" port="4050" protocol="http"/>
  </ClusterAddress>

    <Server CloneID="WSESRV.WSESRVA" Name="SC48">
      <Transport Hostname="202.5.10.4" Port="4050" Protocol="http"/>
    </Server>

    <Server CloneID="WSESRV.WSESRVB" Name="SC50">
      <Transport Hostname="202.5.10.6" Port="4050" Protocol="http"/>
    </Server>

```

```

<Server CloneID="WSESRV.WSESRVC" Name="SC52">
    <Transport Hostname="202.5.10.7" Port="4050" Protocol="http"/>
</Server>

</ServerGroup>

<!-- A route ties together each of the above components. -->

<Route ServerGroup="FMISrvr" UriGroup="default_host_URIs"
VirtualHostGroup="Franck"/>
<Route ServerGroup="FMXSrvr" UriGroup="default_host_URIs"
VirtualHostGroup="FranckTest"/>
<Route ServerGroup="INTSrvr" UriGroup="default_host_URIs"
VirtualHostGroup="WilyTrade"/>
<Route ServerGroup="OMESrvr" UriGroup="default_host_URIs"
VirtualHostGroup="CandleTrade"/>
<Route ServerGroup="WSMSrvr" UriGroup="default_host_URIs"
VirtualHostGroup="CyaneaTrade"/>
<Route ServerGroup="INXSrvr" UriGroup="default_host_URIs"
VirtualHostGroup="WilyeRWW"/>
<Route ServerGroup="OMXSrvr" UriGroup="default_host_URIs"
VirtualHostGroup="CandleeRWW"/>
<Route ServerGroup="WSXSrvr" UriGroup="default_host_URIs"
VirtualHostGroup="CyaneaRWW"/>

</Config>

```

When the plugin receives an HTTP request for consideration, it goes through its Route statements trying to match the input URL with a target Server Group. The input is identified in one of two ways:

1. From the URI part of the URL, if there is a match with a URI definition. In our case we inserted the wildcard definition `/*` to make sure that all URIs were accepted.
2. From the host part of the URL, if there is a match with a Virtual Host definition. We defined a virtual host for each of the eight port numbers in which we were interested.

Once a server group has been identified from the port number, the plugin has one more choice to make. It checks the incoming request for the JSESSIONID keyword in any attached cookie:

- ▶ If there is no such keyword, then there is no affinity with any previous HTTP request. The plugin sends the request to the IP address and port defined in the ClusterAddress statement.

- ▶ If there is such a keyword and it matches one of the ClonelDs in the Server statements, then there is an affinity with the server instance of that name. The plugin sends the request to the IP address and port defined in the appropriate Server statement.

We defined our cluster addresses to be the distributed VIPAs in the sysplex, and our individual server addresses to be the static VIPAs in the appropriate LPARs. We had to use two distributed VIPAs because of the (soon to be fixed) restriction that each distributed VIPA can be associated with no more than four ports.

B.2 z/OS TCP/IP definitions

Example B-3 is part of the TCP/IP profile in the “test” stack on SC48 (in other words, the stack connected to our test network).

Example: B-3 SC48 TCP/IP profile

```

IPCONFIG DATAGRAMFWD VARSUBNETTING SYSPLEXROUTING
  DYNAMICXCF 192.168.80.4 255.255.255.0 4
;
VIPADYNAMIC
VIPADefINE MOVE IMMEDIATE 255.255.255.0 202.5.10.10 202.5.10.11
VIPADefINE MOVE IMMEDIATE 255.255.255.0 202.5.10.12
VIPADISTRIbUTE DefINE 202.5.10.10 PORT 900 5555 8080 1389
  DESTIP 192.168.80.4 192.168.80.6 192.168.80.7
VIPADISTRIbUTE DefINE 202.5.10.11 PORT 4040 5050 6060 7070
  DESTIP 192.168.80.4 192.168.80.6 192.168.80.7
VIPADISTRIbUTE DefINE 202.5.10.12 PORT 4050 5060 6070 7080
  DESTIP 192.168.80.4 192.168.80.6 192.168.80.7
ENDVIPADYNAMIC
... ..
DEVICE SVIPA VIRTUAL 0
LINK LVIPA VIRTUAL 0 SVIPA
... ..
HOME
  10.1.6.1 OSA2100LNK
  202.5.10.4 LVIPA

```

Note the relationship between the addresses defined in VIPADefINE and VIPADISTRIbUTE, the ports defined in VIPADISTRIbUTE, and the ClusterAddress statements in Example B-2 on page 292. A TCP connection request received on a matching address/port pair will be distributed to the stacks identified in the DESTIP keyword in the same statement.

Note also the correspondence between the static VIPA address 202.5.10.4 and the addresses for the xxxSRVA server instances in Example B-2 on page 292. The instances running on SC48 all have the instance name xxxSRV.xxxSRVA.

SC48 is the primary distributing stack. On SC50 and SC52, our VIPADYNAMIC block contained no VIPADefine or VIPADISTRIBUTE statements for the two addresses. It had only VIPABACKUP statements, as SC50's example in Example B-4 shows.

Example: B-4 SC50 TCP/IP profile

```
IPCONFIG DATAGRAMFWD VARSUBNETTING SYSPLEXROUTING
  DYNAMICXCF 192.168.80.6 255.255.255.0 4
;
VIPADYNAMIC
VIPABACKUP 80 202.5.10.11
VIPABACKUP 100 202.5.10.12
ENDVIPADYNAMIC
... ..
DEVICE SVIPA VIRTUAL 0
LINK LVIPA VIRTUAL 0 SVIPA
... ..
HOME
  10.1.6.3 OSA2100LNK
  202.5.10.6 LVIPA
```

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 302.

- ▶ *Enabling High Availability e-Business on zSeries*, SG24-6850
- ▶ *WebSphere for z/OS V4 Problem Determination*, SG24-6880
- ▶ *DB2 Performance Monitor for z/OS*, SG24-6867
- ▶ *IMS Version 7 Performance Monitoring and Tuning Update*, SG24-6406
- ▶ *IBM Tools: CICS Performance Analyzer V1.2*, SG24-6882
- ▶ *CICS Transaction Gateway V5 The WebSphere Connector for CICS*, SG24-6133

Other resources

These publications are also relevant as further information sources:

- ▶ *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834
- ▶ *WebSphere Application Server V4.0.1 for z/OS and OS/390: Operations and Administration*, SA22-7835
- ▶ *WebSphere Application Server V4.0.1 for z/OS and OS/390: Assembling J2EE Applications*, SA22-7836
- ▶ *System Management User Interface*, SA22-7838
- ▶ *HTTP Server Planning, Installing, and Using*, SC34-4826
- ▶ *z/OS V1R3.0 MVS System Management Facilities (SMF)*, SA22-7630
- ▶ *z/OS V1R2.0 MVS Workload Management Services*, SA22-7619
- ▶ *z/OS V1R2.0 MVS Programming: Resource Recovery*, SA22-7616
- ▶ *z/OS Resource Measurement Facility User's Guide*, SC33-7990
- ▶ *z/OS Resource Measurement Facility Report Analysis*, SC33-7991

- ▶ *z/OS Resource Measurement Facility Performance Management Guide*, SC33-7992

Referenced Web sites

This Web site is also relevant as further information source:

- ▶ WebSphere Application Server for z/OS and OS/390 support page:
http://www-3.ibm.com/software/webservers/appserv/zos_os390/support.html

How to get IBM Redbooks

You can order hardcopy Redbooks, as well as view, download, or search for Redbooks at the following Web site:

ibm.com/redbooks

You can also download additional materials (code samples or diskette/CD-ROM images) from that site.

IBM Redbooks collections

Redbooks are also available on CD-ROMs. Click the CD-ROMs button on the Redbooks Web site for information about all the CD-ROMs offered, as well as updates and formats.

Index

A

- Activity Records 64
- address space 6
- All Workloads workspace 163
- analyzing accounting data 54
- APPL% 50, 80, 85
- APPLENV 77
- application activity cycle 42
- Application Activity Display 254
- Application Monitor 228
- Application server model 11
- Application trace 164
- application-centric 227
- application-specific data caches 115
- application-specific error conditions 115
- ARM 89
- automated monitoring 227
- Automatic Restart Manager (ARM) 89
- AVG ENC 49

B

- BBOC_HTTP_TRANSACTION_CLASS 36
- Bean-Managed Persistence 10
- Blame Technology 114

C

- Candle Management Server (CMS) 161
- CandleNet Portal Server (CNPS) 161
- capacity planning 29
- Channel Subsystem 4
- Choose Server 251
- CICS 115, 126, 170
- CLASSPATH 21
- component trace options 76
- Configurable Monitoring Level 227
- Container-Managed Persistence (CMP) 10, 97
- containers 9
- control region 12
- CORBA 114
- Coupling Facility 5, 169
- CPU Activity Report 43
- CPU report 40

- Current Situation Values table 182
- CustomerEntityBean 129, 141
- Customizable security 227

D

- dashboard, free-form 116
- Data Archival 233
- Data Collectors 228
- Data Set Parameters 274
- DB2 97–98
- DB2 accounting data 54
- DB2 accounting times 54
- DB2PM 54
- DBC 8
- debugOut 135–136, 217, 220, 274, 277
- directive files 114, 124
- Dispatch Time Data Effective 45
- drill-down
 - approach 235
 - methodology 227

E

- ebusiness navigator tree 177
- eITSO 98
- EJB 114
- EJB Coverage box 256
- EJB Method Summary 252–253
- EJB Summary 251
- ejbFindByPrimaryKey 197
- ejbFindByPrimaryKey 129
- ejbFindCustomerByLastName 141, 176–178
- enclave 48, 82
- Enterprise Manager 116
- entity bean 9
- Environment Performance Agent 112
- eRWW 98
- Example 1 100, 140, 173, 237
- Example 10 101, 134, 148, 217
- Example 11 153, 220
- Example 2 100
- Example 3 100, 136, 181, 243
- Example 4 100, 126, 188, 245
- Example 5 100

Example 6 100, 129, 194, 253
Example 7 100, 144, 200, 260
Example 8 101, 145, 207, 264
Example 9 101, 212, 268
Examples 99, 126, 173
 summary 101
 used 100

F

findCustomerByLastName 259

G

garbage collection 37
garbage collection (GC) 67, 186
Global Performance Management Control 43

H

hardware and software components 89
Hardware Management Console (HMC) 43
HFS 168, 208, 210
hierarchical file system (HFS) 6
hit rate 27

I

IBM HTTP server recommendations 32
IFASMFDP 58
IN READY 48
In-Flight Request Search 260
instrumentation 114
instrumentation points 114
Interface Repository (IR) servers 15
Interval Records 64
Introscope 110
 agents 111
 Enterprise Manager 111
 Environment Performance Agent 115
 Explorer 141
 Major Components 111
 Workstation 111
ITSO Configuration 123, 236

J

J2EE 8
 components 9
 Container 10
 deployment descriptor 11
JAF 8

Java

Agent 112, 125
Connector Architecture (JCA) 9
DataBase Connectivity (DBC) 8
Message Service (JMS) 8
Naming and Directory Interface (JNDI) 8
RMI 114
Transaction API (JTA) 8
Java Virtual Machine 21
JavaBeans Activation Framework (JAF) 8
JavaMail 8
JCA 9, 115, 170
JDBC 114, 170
JDBC tracing 76
JMS 8, 163, 170
JMSController 250
JNDI 8, 163
JRAS tracing 76
JSP 97, 162
JTA 8, 115
JVM
 garbage collection 186
 heap size 69, 166, 187
 I/O 114
 Memory Usage 244
 profiler interface (JVMPi) 166
JVM_ENABLE_VERBOSE_GC 67
JVM_HEAPSIZE 69
JVMPi 166

L

LDAP 21
Leak Hunter 120, 122, 138
logging 134
logical partition (LPAR) 4
Longest Running Workloads 175
LPAR 5
 BUSY TIME 47
 cluster 46
LPARCE
 Capacity Estimator 46

M

managed-object framework (MOFW) 165
MAX_SRS 34
measurement interval 41
memory leak 67, 69, 82, 122, 134, 136, 181, 187, 243

message bean 10
Message Queueing (MQ) 22
MIN_SRS 34
MOFW 165
monitoring console 228, 236
MQ Series 115
MVS BUSY TIME 47

N

node 14
normal monitoring mode 114
NOT ACCOUNT time 56

O

observe the current memory usage 244
OMEGAMON XE
 agents 161
 architecture 158
 Candle Management Server 161
 CandleNet Portal Server 161
 CICS 170
 clients 159
 DB2 170
 DB2plex 170
 IMSpIex 170
 OS/390 167
 OS/390 UNIX System Services 168, 173
 performance monitors 158
 WebSphere Application Server 165, 173
 WebSphere MQ 172, 201
OMEGAMON XE for DB2plex 200
Open System Adapter (OSA) 89

P

package 7
Page View Rate 27
paging activity 74
Parallel Sysplex 5
partition data report 40
PathWAI configuration 170
PathWAI Dashboard for WebSphere Infrastructure 170
performance
 analysis 29, 42
 configuration guidelines 31
 expectations 28, 31, 72
 metrics 116

 monitoring 29
Physical Management Time 44
ProbeBuilder 114
processing weights 46, 167
PRR 98
Publish Traffic 232

R

Redbooks Web site 302
 Contact us xii
Remote Method Invocation - Internet Inter Orb Protocol (RMI-IIOP) 8
Report Filtering Options 272, 274
Request Detail 261
Resource Recovery System (RRS) 89
response time 26
 alert 123, 125, 173
 distribution 52
 expectations 83
 objective 37
RMF
 CPU information 40
 partition data report 43
 Post Processor 42
 reports 39, 42
RMI-IIOP 8
RRS 89, 169
RRS Attach Facility (RRSAF) 20

S

Selected Workload - History workspace 183
Selected Workload Occurrences view 184
Server and Report Type Selection 246
Server Availability Detail 244, 254
server instance 12
server region 12
Server Region address space classification 37
Server Region enclave classification 35
servlet 97
session
 affinity 97
 bean 10
 EJB 97
SimpleFileServlet 63, 146, 207
SMEUI 15
SMF
 Record Interpreter 57, 60
 record type 120 63, 165

- records 39
- type 120 231
- Snapshot Traffic 233
- Software Consistency Check 245
- SSCHRT 49
- stalled request alert 125
- stateless EJB 98
- STD DEV 50
- STORAGE 52
- subtask 12
- summary report 40
- sysplex configuration 88
- Sysplex Distributor 89, 94
- Sysplex Timer 5
- System Administration model 15
- System Management Facility (SMF) 39
- System Properties 254
- System Resources Overview 251, 254
- Systems Management End User Interface (SMEUI)
15

T

- test workloads 96
- Think Time 28
- thread 12
- throughput 26
- To evaluate the overall health of the server farm
268
- Trade2 97, 163, 246, 251
- transaction 27
 - class 36
 - trace mode 114
- Transaction Tracer 120, 128–129
- Trap and Alert Management 235, 254
- Trend Report 248, 274

U

- UNIX System Services 6, 98, 168
- user-defined threshold 116

V

- verbose GC trace 67, 83
- View Method Trace 277

W

- Web container 97
- WebSphere

- class loaders 114
- SMF Record Interpreter 57
- Studio Application Monitor 226
- Studio Workload Simulator 29, 98
- Studio Workload Simulator (WSWS) 93
- WebSphere Edge Server 94, 207
- Whole Application View 110
- WLM queues 35
- Workload Activity Report 48, 85
- Workload analysis 163
- Workload Manager 6, 169
- Workstation
 - Console 116
 - Console Editor 116
 - Explorer 116
- WSAM
 - Problem Determination mode 230
 - Production mode 230
 - Profiling mode 231
- WSWS 93

X

- XCF 5
- XML 114
- XSL 114



Monitoring WebSphere Application Performance on z/OS

(0.5" spine)
0.475" <-> 0.875"
250 <-> 459 pages



Redbooks

Monitoring WebSphere Application Performance on z/OS

**Monitor and
troubleshoot
production
performance of
WebSphere on z/OS**

**Introscope, PathWAI,
WebSphere Studio
Application Monitor**

**WebSphere on z/OS
performance
methodology**

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 or OS/390 performance management in order to ensure the successful deployment of e-business applications.

This redbook helps you understand WebSphere Application Server V4.0.1 performance factors, and how you can monitor your system and application performance, response time, and resource utilization.

Part 1 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. Examples are given to explain how to narrow down to the source of the problem. Interpretation of data and rules of thumb are provided.

Part 2 expands on performance monitoring products available for WebSphere on z/OS that will help detect and identify performance problems:

- Candle Corp. PathWAI Dashboard for WebSphere Infrastructure
- IBM WebSphere Studio Application Monitor
- Wily Technology Introscope

For each product, we describe the relevant methodology and show, through typical real-life examples, what to look for in determining where the performance bottleneck is.

**INTERNATIONAL
TECHNICAL
SUPPORT
ORGANIZATION**

**BUILDING TECHNICAL
INFORMATION BASED ON
PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:
ibm.com/redbooks**