IBM Press Program Manager: Tara Woodman, Ellice Uffer
IBM Press Consulting Editor: Surekha Parekh
Cover design: IBM Corporation
Published by Pearson plc
Publishing as IBM Press

*Library of Congress Catalog Number*
***2004111096***



ISBN 0-13-185587-5

Text printed in the United States on recycled paper at *R.R. Donnelley & Sons in Crawfordsville, Indiana*.
First printing, *December, 2004*

# Foreword

When we first introduced the IBM® WebSphere® Application Server software (WebSphere), we had a vision that we were creating *the* foundation for information computing for the next several decades. We had as inspiration the IBM traditions of TPF, IMS, CICS®, DB2® and MQ—middleware platforms that have been hosting mission-critical business applications for the last 40 years. Only the future will tell whether our aspirations will be realized but in the short history that has amassed so far every indication is that we have established a solid foundation for achieving exactly that. WebSphere is used at some of the largest corporations in the world for both web-based information-computing as well as core on-line transactional computing. Companies such as eBay, Bank of Tokyo-Mitsubishi, Daimler-Chrysler Corporation, Dresdner Bank, Office Depot, SAS Airlines, Toshiba, Siemens Medical Solutions, as well as thousands of other customers use WebSphere to run their businesses.

At the end of 2003 we saw WebSphere take a commanding lead in the Java® 2 Enterprise Edition (J2EE) Application Server marketplace—somewhere in the vicinity of 29%[1] to 41%[2] market-share, depending on how you count it. WebSphere has gained market-share over BEA, Oracle, Sun or any other J2EE vendor. More importantly, the basic style of computing supported by WebSphere as defined by the J2EE specification is rapidly growing in importance for mission-critical business application deployments, representing a US$2.8B opportunity in 2003[3] just for the middleware to support these environments.

But great stories like these are not just created from marketing charts—they result from the culmination of an understanding of the marketplace, deep insight and experience in computing technologies, a sharp and detailed long-term vision for what information computing can offer to users and, at the end of the day, a great deal of hard work and perseverance—a dedication and passion for pursuing the right things.

As you may be aware, IBM is a large and globally distributed company with development organizations in dozens of cities throughout the world. This is one of the things that makes IBM a great company—it allows us to maintain a high degree of cultural diversity. This, in turn, ensures

[1]IDC, May 2004
[2]Gartner Dataquest, May 2004
[3]Gartner Dataquest, May 2004

that we retain a strong sensitivity to the needs of different users operating under different economic and environmental conditions, and to remain tuned to the patterns of local concerns throughout the world. However, that same diversity can also create huge challenges to development processes. Diversity of culture also means diversity of opinions and perspectives and those differences show up in the technical debates about what's important and why.

Diversity of locale means operating over different time zones, and without the inherent benefits of face-to-face communication. It means working through different holiday and vacation schedules. It means having to matrix across different organizational, legal, geo-political, and sometimes even language barriers. It means forging a common and mutual understanding where none might occur naturally.

When we started the WebSphere project we had a choice between developing the entire WebSphere offering at one location and developing it in a distributed fashion across multiple development sites. The decision was not hard to make. We knew that WebSphere would have to incorporate many different technologies—transaction management, state persistence and query, distributed communication, security, administration, web processing, caching, workload management, messaging, resource management, multi-tasking and contention management, session management, serviceability, high and continuous availability, performance and monitoring, and on and on. We chose to leverage the skills, knowledge and deep histories that already existed within IBM, at the locations that major in each of those technologies. We tapped messaging resources from the MQ team in Hursley, England; query resources from the DB2 team in Santa Teresa (now Silicon Valley Labs), CA; administration resources from the Tivoli team in Austin, TX; integration and componentization resources from the OS/400® team in Rochester, Minnesota; tooling resources from the VisualAge® and now Rational® team in Toronto, Canada; web processing and performance resources for the Servlet Express team in Raleigh, NC; scalable systems resources from the zOS® team in Poughkeepsie, NY; transaction management from the Encina® resources in Pittsburg, PA and CICS resources in Hursley; etc. Sure, that made managing the team and product more difficult (*a lot more difficult!*), but the alternative would have meant reinventing all of that knowledge and skill and that was something that we did not want to do.

As I said earlier, building a successful product like WebSphere takes a lot of work. Going essentially from scratch in 1999 to WebSphere V5 in 2004 has required painstaking refinement of technologies to get them to blend well together, to meet the latest and (at first) rapidly evolving industry standards, to balance and tune the disparity of workload types that different customers experience, and to achieve both something that is easy to install and use and something that will scale up to the largest of installations. We have done that in WebSphere by keeping to a few basic architectural principles, including: *maintaining a clean separation of concerns*—don't let one set of issues coerce a different set of issues; *minimizing the amount of functional duplication*—it is better to put more focus on getting one really good implementation than many poor implementations; on the other hand, *not assuming one-size-fits-all*—use the separation of concerns principle to differentiate quality-of-service and scale differences for different users of the same function.

All of this, then, is set to a more fundamental backdrop. First, WebSphere is a *machine*—it is an aggregation of functional parts that work together as a tool to help you get your job done

more efficiently than if you had to do all the work yourself. Like all machines, the elegance and utility of WebSphere can be measured by how well we have blended the characteristics of *strength*, *precision* and *tolerance*—strength to hold up against the most demanding workloads; precision to execute accurately, securely and efficiently every time it is used; and tolerance to adapt to a wide variety of conditions and uses.

Second, the world is not static. WebSphere needs to support the extremes of usage scenarios, but just as important, customers must be able to transition between different usage scenarios with as little friction as possible—*minimizing the friction of movement*. You may start out with a small web site but, over time, as your business grows so will the demands of your applications—both in terms of the capacity of the underlying hosting environment as well as the sophistication of what you need to do. Other customers will start out with simple tightly integrated business functions, but later will grow to en-compass their entire business functionality across an enterprise service bus. WebSphere needs to be able to facilitate these transitions easily.

Third, the computing system will be touched by many different roles—end-users of the application, deployers, developers, monitors, administrators, service centers, etc. Each of these roles will develop their own, somewhat independent, perception of the utility of the middleware in achieving their goals and responsibilities. WebSphere must optimize the productivity experience of each of these roles.

Fourth, modern distributed computing systems are fundamentally and inherently complicated. Distributed systems combine a large number of moving parts with fragile and inefficient transitional boundaries. This is like putting horse-drawn buggies, freight trucks, and Formula-1 race cars all on the same highway and then managing the entire thing to ensure everything and everyone gets to where they need to go, does what needs to get done, and does so safely and reliably. WebSphere must be able to manage this complexity, and mask out as much of that complexity as possible.

The corollary to this is that the middleware is responsible for virtualizing the infrastructure through an abstraction model that lets businesses focus on their business domain expertise and adding value to their business. Without middleware, we often see application developers spending tremendous amounts of their time dealing with issues like connectivity to data systems, including recovering from connection failures; managing state caches, including handling invalidations; protecting access to information and processes, including administering access policies; and on and on—none of which has anything to do with taking orders, depositing checks, checking inventory, tracking shipments, managing risk, and so forth. The responsibility of WebSphere is to manage the distributed system on behalf of the application—freeing application developers to focus their attention on other, more valuable things.

Fifth, to expand the macro-economics of computing we must maintain a high degree of consistency in the fundamental aspects of information computing. As an analogy, the macro-economics of transportation depends on there being more similarity between trucks than differences. Trucks have the same basic limitations on height, length and width which enables them to fit on the same road structure; they use essentially the same driving configuration which enables a larger pool of skilled drivers; they use the same fuel formulae which enables a broader distribu-

tion channel for power; they use similar engine technologies which enables a large support structure; they conform to a uniform set of deck height assumptions which enables interchangeable loading and terminal implementations. That's not to say that there aren't differences between Mack trucks and Volvo trucks, but those differences are primarily around quality of service and scale differences, not basic functional differences. Said a different way, WebSphere must conform to a set of broad industry standards. Those standards enable the expansion of the marketplace for application serving middleware, and more importantly for enabling customers to derive value from their investment in applications designed to work on WebSphere.

The final backdrop to the principles of WebSphere architecture has to do with the extended nature of information computing. WebSphere is a hosting environment for J2EE-based applications. But WebSphere is also the foundation of a much broader platform for information computing—including portal-serving, business integration, service- and event-oriented architectures, service buses, client interaction and collaboration services. In this capacity, the application server defines the fundamental execution model for the entire platform—the integrity model, the protection model, the availability model, the serviceability model, etc. WebSphere is, in some sense, the network operating system for the information system.

This book is a microcosm of WebSphere itself. Ann, Mike Everett, Mike Casile, Keith, Alan, Kyle, Dave, Tom, Yu, Dipak, Michel, Mihaela, and Radhika all exemplify exactly the kind of dedication, expertise and diversity that you can find throughout the WebSphere development and execution team. They hail from many different locations within IBM, and, like the rest of the WebSphere team, have had to orchestrate and manage the development of this book across timezones and organizations.

The result, however, has been nothing short of outstanding. This book covers the full breadth of the operational experience with WebSphere—from a basic overview of the product through to configuration, deployment, administration, tuning and troubleshooting. Like WebSphere, this book represents a tremendous effort—especially so when you consider that all of the authors have real jobs to attend to through the day. I have absolutely no doubt that you will find this book to be hugely informative—a great aid to getting the very best from your WebSphere runtime.

I truly appreciate the effort these folks have made in producing this book. Just as importantly, I know I speak for the entire team when I convey how enormously grateful we are to the many, many customers, administrators and developers that have embraced WebSphere; that have discovered the value of WebSphere and have come to exploit WebSphere for the many benefits it has to offer. For that we thank you.

**—Rob High, IBM Distinguished Engineer;**
**WebSphere foundation, Chief Architect; and**
**Member, IBM Academy of Technology**

CHAPTER 23

# WebSphere Performance Tuning–z/OS

## Objectives

This chapter covers the following concepts:

- Identify the measurements and perspectives of performance
- Optimal Testing Techniques
- Tuning impacts of WebSphere Configurations and Topologies and z/OS subsystems
- Impact of z/OS subsystems and Java
- Using the monitoring tools for WebSphere and z/OS and knowing which to apply to different types of problems

## 23.1 Overview

The purpose of this chapter is to assist you in tuning the performance of WebSphere V5 on your z/OS system. The general focus will be on z/OS-specific items, as the previous chapter discussed more general information. We mention tuning here as performance is not something to be considered only as an afterthought or when the application is perceived as not performing well (performance problem). These are certainly times when performance does need more focus, but performance should be considered from the earliest stages of any product life cycle.

z/Linux performance considerations are covered in Appendix H (z/Linux).

### 23.1.1 Problem versus Perception and z/OS Resources

When someone perceives and reports a performance problem, it is important to understand the basis of the performance expectations relative to the application and the resources devoted to it. Applications running in a J2EE environment tend to take more resources (specifically memory and CPU) than typical legacy applications running on z/OS. If all tuning has been done correctly,

it may be that the application simply requires more resources than were previously estimated. This often happens when the application is modified and becomes more complex than originally intended. You must also understand the z/OS resources "assigned" to your application.

It is beyond the scope of this document to discuss the cycle times or processor power of the z/Series hardware configurations, but there are some important things to note:

- z/OS CPU resources tend to be shared. That is, your image/Logical Partition (LPAR) is usually given some share or percentage of the total CPU resources available. This gets complicated further when you consider that LPARs can take advantage of unused cycles from other LPARs if they have shared CPUs and are not capped. A good Resource Management Facility (RMF) report can yield much information on this.
- Your z/OS LPAR or image will often be running WebSphere and one or more subsystems (DB2, CICS, IMS, etc.). Thus, a direct comparison between WebSphere on the distributed platforms using these back-ends and WebSphere on z/OS using them must account for the extra CPU cycles used by the back-end system itself.
- z/OS systems generally run a mixed workload and are not dedicated to WebSphere applications as are servers on distributed platforms. This makes it more important to understand the impact that WebSphere is having on the other workloads and not just the overall system performance.
- The importance of having sufficient memory (discussed later along with paging) for the workload on the system cannot be overstated. There is little marginal cost for memory and having insufficient memory can cause paging, which degrades performance.

A common issue with z/OS resources occurs when someone tests an application on their workstation (e.g., a uniprocessor 2.4 GHz machine) and then moves it up to z/OS (the "big" box). z/OS images for test or development are often given a very small share of a full z/OS machine. We have seen images running as little as 150 MHz. Thus, instead of achieving a great performance boost by moving to z/OS, performance actually degrades.

WebSphere z/OS, in nearly all cases, achieves equivalent performance to WebSphere on the distributed platforms when comparing the same amount of CPU resource. On the other hand, z/OS hardware tends to be more expensive than distributed platforms. The extra cost of z/OS hardware is offset by its qualities of service (QOS) which are superior to other platforms. It is generally a balancing act between QOS and cost as to how an application is deployed. This discussion is beyond the scope of this chapter.

## 23.2 Repeatable Performance Scenarios

### 23.2.1 Test System

In any performance tuning scenario, it is best to have a "controlled environment" such that running the same test multiple times will result in the same (or nearly the same) outcome. In addition, it should be a test system where running tests, recycling servers, modifying configurations,

or adding heavy traces does not impact production. This isolates the impact of your changes to the change itself and reduces "noise."

This test system should also be similar to the production system so that it can be used to accurately predict the performance of the production system. Items here would include software levels, access to enterprise information servers (DB2, CICS, IMS, etc.), memory, and so on. The actual CPU resource of a test system is normally smaller than the production system.

### 23.2.2 Test Tools

There are many excellent test tools that can be used to generate sample browser workloads. These tools are ideal for performance work as the same scripts (same requests, same number of concurrent users, and same wait time) can be run repeatedly. We do not review individual tools available in this document, other than to say that most of them can adequately drive load against your system.

#### 23.2.2.1 Scripts

When using the test tools, it is best to set up test scripts that resemble a particular set of functions in the application. The simpler scripts may yield more predictable behavior and isolate smaller levels of functionality. These are ideal, especially if a problem is occurring. For example, you might want to create a test script for each individual transaction and capture performance data with multiple users executing the same transaction. You might also want to have a test script with multiple users concurrently executing all transactions so you can study the interactions of your transactions

If it is feasible to create a representative workload without making updates to the database, this is an ideal situation. If this is not the case, then the back-end data store needs to be periodically reset.

#### 23.2.2.2 Restoring the Data for Each Test

Resetting the data store in DB2 is the most common requirement. This can be done via various third party tools or simply by unloading and reloading the specified tables. The unload and reload methods run quickly and reset the database to a consistent state before each test. While the unload need only occur once (when the database is in an initial state that is ideal for the test), the reload should run before each test. It is a good idea to also call run RunStats after each reload (can be an added step in the reload job). Resetting other resources is beyond the scope of this chapter.

### 23.2.3 Simplification of Scenario

It is best to have the simplest possible scenario to re-create any situation on the system. Applications can become complex, including back-end data stores or execution environments, and network infrastructure, including firewalls, and so on. Initial performance testing should include as much of this complexity as is feasible. If a workload does not meet expectations, it is often best to break it down into the simplest component parts feasible. This can isolate the problem component(s), and it makes it easier for any external support to re-create and work on the problem.

**TIP**
Using test tools placed close to the system under test (SUT), it is often a simple procedure to eliminate firewalls, Web servers, and other network components, thus isolating the work in the J2EE server and the back ends.

## 23.3 Relevance of z/OS and Subsystems

In order to provide reliability and other qualities of service (QOS), z/OS has many subsystems, and the WebSphere workload often requires modifications to the configurations of these subsystems. An excellent source of information on this tuning can be found in chapters 9 and 10 of the *WebSphere Application Server for z/OS V5 Operations and Administration Manual* at the following URL: *ftp://ftp.software.ibm.com/software/webserver/appserv/zos_os390/v5/bos5b1001.pdf.* Below is a brief mention of several of the key subsystems and their roles in keeping the WebSphere container running well. All of them should be kept current on maintenance, and component (and system) tracing should be turned off or minimized.

### 23.3.1 Transmission Control Protocol (TCP)

Transmission Control Protocol (TCP) and the network are critical to the response time perceived by the customer when doing work on z/OS. It is best to have a fast network adapter (e.g., Gigabit OSA) and to periodically monitor its performance. If tuned well, TCP should account for only a small amount of the total response time (less than 2 percent) experienced by the customer. Firewalls are required in any production environment, but, again, tuning these is critical.

**TIP**
The resource utilization of the TCP address spaces can be determined with WLM and RMF. It may also be a TCP issue if the controller address space is taking more than 1 percent CPU. To understand the low level details of TCP flows, your TCP Systems Programmers can run a filtered packet trace.

### 23.3.2 UNIX System Services (USS)

USS is the environment where WebSphere runs. System traces can reveal a great deal about the behavior of the application and the J2EE server. If misused, however, they can more than double the path-length of some requests. A good reference for USS tuning can be found at

*http://www-1.ibm.com/servers/eserver/zseries/zos/unix/perform/webtun.html.*

### 23.3.3 Resource Recovery Services (RRS)

Resource Recovery Services are used extensively by WebSphere on z/OS to handle transactional behaviors and access to all recoverable resources. There are numerous configuration options here that can impact the operation of RRS. One key option is using a Coupling Facility (CF) logstream

to avoid disk IO. Optimizations in this area have made database two-phase commit processing on z/OS notably faster than on any other platform. A few key items to watch in RRS include

- Sizing the RRS logs so that they are not offloading too often or overflowing the CF
- Keep Main and Delayed logs in the CF
- Only use the archive log if absolutely needed (to do PD on an unstable system)

**TIP**
The most common RRS related issue is log size. WebSphere can increase the amount of data logged and create a need for larger logs. Issues here can generally be seen in the z/OS System log with an increase in log switching and archiving. If this is occurring, then the RRS people may need to resize the logs. System Logger accounting data can be found in SMF Type 88 records. Sample program IXGRPT1 in Sys1 .SAMPLIB shows how to produce a report using SMF Type 88 records.

### 23.3.4 Cross-System Coupling Facility (XCF)

The Cross-System Coupling Facility is less important to WebSphere V5 than to WebSphere V4. In V4, servers with instances on different images within a SysPlex had to use a shared HFS, which can cause intensive XCF activity. V5 no longer has this requirement. If shared HFSs are being used, however, it is important that the owning image be the one that does the most IO (especially output) to the HFS.

**GOTCHA**
While Shared HFSs can be a great means of communication and centralization of control, improper use can have a severe performance penalty. Using them for logging can be a particularly expensive practice.

**TIP**
While the first z/OS image (LPAR) to mount a shared hfs becomes the owner, there are dynamic means of changing ownership. This is discussed in the UNIX System Services Command Reference (SA22-7802).

**TIP**
A view of the XCF service class in RMF can help to isolate excessive CPU time being used by XCF. If the XCF address space is taking more than 4 percent of a CPU engine, a USS Systems Programmer should look more closely at XCF (it may or may not be WebSphere related).

### 23.3.5 Workload Manager (WLM)

The z/OS Workload Manager differentiates WebSphere z/OS from the distributed platforms. On WebSphere for z/OS, a server consists of a Controller address space and one or more Servant address spaces (discussed later in the Topology section). WLM can be used to control the number

of Servants to meet goals defined for the various workloads. WLM is discussed in chapter 19, Workload Management Overview: z/OS.

WLM can also balance workloads across clustered servers in certain situations (although HTTP workload distribution is generally driven by the front-end mechanisms [such as the IHS plug-in or Edge Servers] and EJB calls, in an optimal scenario, will not leave the server).

In the section on RMF and WLM, we discuss classifying work and creating reporting classes so that the total cost of handling transactions can easily be determined.

**TIP**

It is advisable to limit the number of different classes of work routed through an individual server. A Servant address space can service only one class of work at any time. This means that many different classes of work will either result in many Servant address spaces, or work queuing up for existing Servants to be reclassified.

### 23.3.6 Miscellaneous Considerations

There are miscellaneous items to keep in mind, including:

- In Resource Access Control Facility (RACF), set BPX.SAFFASTPATH on. This makes security checks in the HFS quicker.
- In WebSphere V5, we recommend that you configure the Link Pack Area (LPA) as recommended during install. This will allow all WebSphere address spaces to share the 40 Mb of load modules instead of each address space loading it into private memory. This is a bit more difficult if you are coexisting with WebSphere V4. Only one of the versions can use LPA, and the other must use STEPLIB DD statements.

## 23.4 The Container

### 23.4.1 Topology

As with all platforms, the J2EE Server z/OS can be run as a stand-alone server, or servers can be incorporated into cells and clustered with other servers. In the V5 release, cells cannot be heterogeneous. Cells must contain only z/OS servers, and they must all be contained within the same sysplex. The restrictions should be eased in future releases. In that there are limitations on distributed platforms (i.e., CTG on distributed cannot fully participate in two-phase commit), there may be compromises in running applications across different platforms.

At a more granular level, there is a difference between a distributed server and a z/OS server. A distributed server is a single process that runs a JVM. A z/OS server consists of two or more address spaces (processes) with different roles, each running a JVM. As shown in Figure 23.1, there is a Controller and one or more Servants. The Controller handles

1. Communications (it is the end point for container communication, including Message-Driven Beans [MDB]). Of course, applications can use HTTP, Java Secure Socket

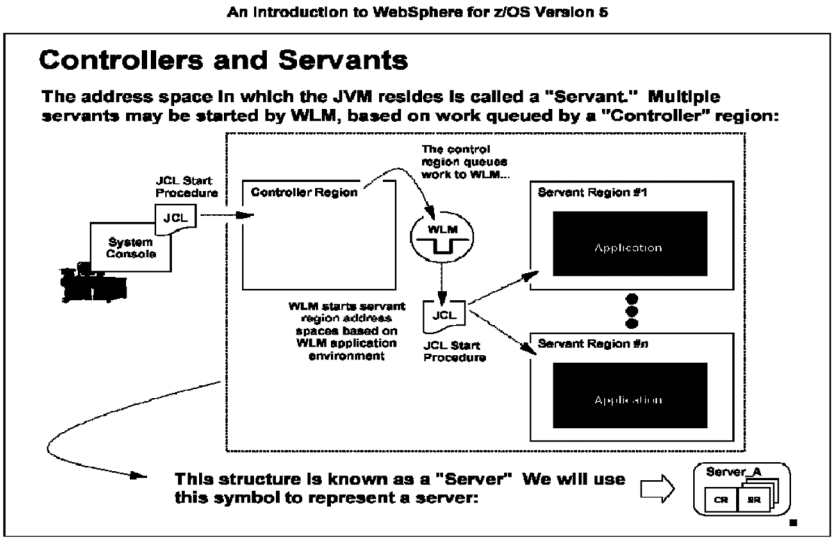


**Figure 23.1** WebSphere z/OS V5 Controllers and Servants.

Extension (JSSE), or other transport clients for specific outbound communication from the Servant.

2. Most Object Requestor Broker (ORB) functionality.
3. Transactional behavior.
4. Security.
5. Executing authorized code.

The Servant is the JVM where the application(s) run. Servants are all on the same image (LPAR). To achieve the parallelism and failover of splitting the work across multiple images, you must set up servers under nodes on each image and cluster them.

Network Deployment is the more appropriate topology for a robust production environment as shown in Figure 23.2. The components in the topology are similar to the distributed platforms, with the following differences:

1. A server is two or more address spaces.
2. A cell can consist only of z/OS nodes and is constrained to a single sysplex.
3. It is more common for z/OS images to contain multiple nodes and for a sysplex to contain multiple cells.

**TIP**

WebSphere V5 can be architected in any number of ways. To accurately predict performance of a production system, the architecture of a test system should be as close as possible to the architecture of the production system.

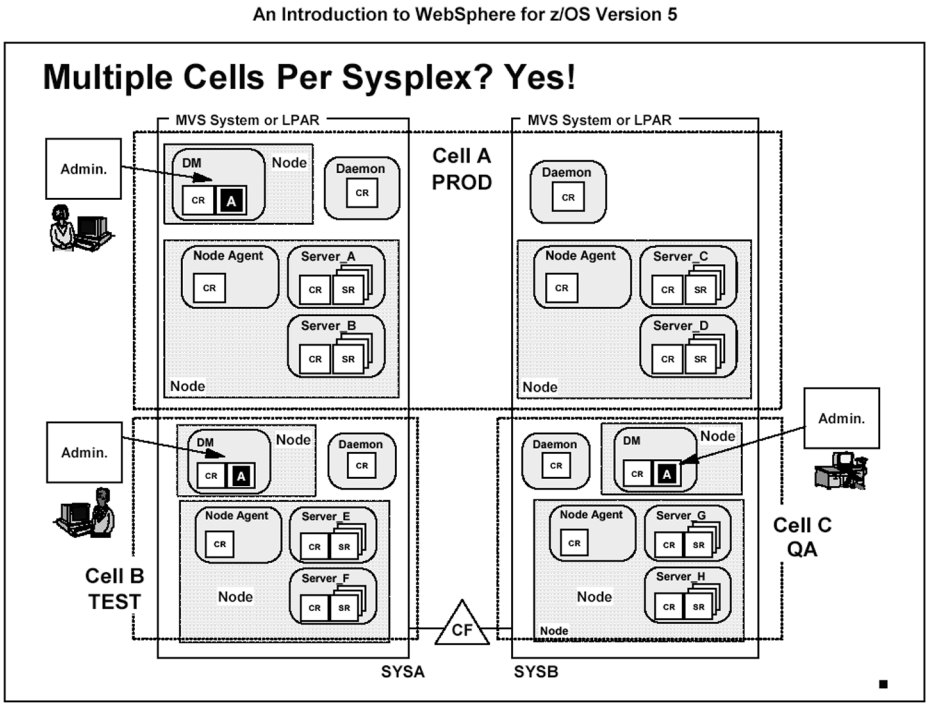


**Figure 23.2** WebSphere z/OS V5 overall topology possibilities.

### 23.4.2 Run Time Settings in the Controller

Many environmental settings have a direct impact on performance. As work arrives at the Controller, it is classified and placed on the appropriate WLM queue (for WLM details, see chapter 19). The WebSphere Administrator can control how the work is classified into Transaction Classes. The WLM administrator can then use these Transaction Classes to classify the work into Service Classes with defined goals. These goals will cause WLM to appropriately prioritize and report on the WebSphere work on the system or sysplex. The WebSphere Controller should run with a high-velocity goal as it is the initial dispatch point to potentially multiple Servants. Servants should also run with a high-velocity goal (not quite as high as the Controller) as dispatching work should not be a bottleneck. Garbage Collection also runs under this class (nonenclave). Prioritization of the enclaves is business-need dependent.

## 23.5 The Java Virtual Machine (JVM)

Each server has a JVM in the Controller and one in each Servant. The focus here is on the JVM in the Servant where the application code executes. This JVM is used by the EJB container and the Web container in the Servant. Understanding the behavior of the JVM as your application executes is critical. If all goes well:

1. Garbage collection (GC) should take 1 to 5 percent of the time
2. There should be no long waits (more than 6 seconds) for stopping the threads.
3. An Allocation Failure (AF) should never cause multiple GCs.
4. Compaction actions should be occurring on less than half of the GCs.
5. The percent free memory should not be reduced over time.
6. Appropriate methods should be Just In Time Compiled (JITted).

### 23.5.1 Garbage Collection

Garbage collection is the process the JVM uses to clean up objects that are no longer referenced. Garbage Collections are started when the JVM receives Allocation Failures or System.gc() calls. An allocation failure is a normal occurrence when there is insufficient free space in the JVM heap to allocate a new object. This causes the JVM to do garbage collection to free up the memory utilized by obsolete objects (objects with no references to them, or garbage).

**TIP**

As a programming best practice, explicit System.gc() calls should be avoided. The JVM has complex algorithms to optimize garbage collection. Placing System.gc() calls in the code causes extra garbage collections that interfere with the JVMs ability to maintain itself. In addition, code may execute more often than expected causing requests for garbage collection to queue up. As this occurs, it can prevent the JVM from getting any real work done.

#### 23.5.1.1 Understanding verboseGC Output

The simplest way to get an understanding of the health of the JVM is to turn on verbose garbage collection (VerboseGC). This can be set via a checkbox in the Administrative Console by navigating to **Servers → Application Server →** ***ServerName*** **→ Process definition → Java Virtual Machine**. Leaving VerboseGC running does not impose much overhead (less than 2 percent) on the system. We recommend having it on for all test systems and even in production when new code is running. The verboseGC output varies from one release of the JVM to the next, but resembles what is seen in Listing 23.1.

**Listing 23.1** Normal Garbage Collection output.

```
<AF[24]: Allocation Failure. need 68376 bytes, 88964 ms since last AF>
<AF[24]: managing allocation failure, action=2 (559688/268368384)>
<GC(30): GC cycle started Wed Jul 30 14:40:55 2003
<GC(30): freed 175851936 bytes, 65%% free (176411624/268368384), in 1642 ms>
   <GC(30): mark: 1447 ms, sweep: 195 ms, compact: 0 ms>
   <GC(30): refs: soft 0 (age >= 32), weak 0, final 89, phantom 0>
 <AF[24]: completed in 1652 ms>
```

**Table 23.1** Breakdown of Garbage Collection Output

| Item | Description |
|---|---|
| <AF[24] | Twenty-Fourth Allocation Failure since the JVM was started. |
| Need 68,376 | Size of the request for memory that caused the failure. |
| 88964 ms | Number of milliseconds since last Allocation Failure. |
| 559688/268368384 | Free heap vs. total heap before GC. Nearly 600K available, but no chunk big enough to alloc 68k bytes. |
| <GC(30) | Thirtieth GC since JVM started (since the twenty-fourth Allocation Failure is causing the thirtieth garbage collection, then either System.gc() is being called or large allocations are causing problems for the JVM (the case here). Either of these conditions can raise a red flag. |
| Freed 39334152 | Number of bytes freed. |
| 65% | Percent of free heap after GC. Memory leaks will cause this number to get progressively smaller and compactions will have less positive impact. |
| 176411624/268368384 | Free heap vs. total heap after GC. |
| 1642 ms | Milliseconds that the GC process took (in a healthy JVM, these can be predominantly sub-second). |
| Compact 0 ms | Compact time (no heap compaction was needed this GC). Compaction occurs when the free space in the heap is fragmented. |
| Refs: soft 0 | No soft references cleaned up here. |
| Age >= 32 | Soft references NOT referenced in 32 GCs will be GC'd. |
| Completed in 1652 ms | Total time taken to resolve allocation failure (usually a small amount, larger than GC time). |

Table 23.1 shows the most value from this listing.

So, in this case it had been 89 seconds since the last allocation failure, and it took 1.65 seconds to resolve this failure. Percent of time in GC is 1652 * 100 / (88964 + 1652), or 1.8 percent. Any time this number climbs above 3 percent, it should be reviewed. It may be expected behavior for some applications to require intense GC processing. Application characteristics that result in GC burning a greater amount of CPU cycles include

1. System.gc() calls
2. Intensive object use and discarding. This can be reduced by:
   a. Use of StringBuffer instead of String
   b. Object reuse and pooling
   c. Allocating arrays and vectors more accurately

3. Allocation of extremely large objects
4. Memory Leaks

A less healthy GC might look like Listing 23.2.

**Listing 23.2** Problem Garbage Collection.

```
<AF[52]: Allocation Failure. need 11825976 bytes, 9216 ms since last AF>
<AF[52]: managing allocation failure, action=2 (74646856/268368384)>
<GC(64): GC cycle started Wed Jul 30 14:50:26 2003
<GC(64): freed 68031744 bytes, 53%% free (142678600/268368384), in 4060
         ms>
   <GC(64): mark: 1631 ms, sweep: 188 ms, compact: 2241 ms>
   <GC(64): refs: soft 0 (age >= 32), weak 0, final 14, phantom 0>
   <GC(64): moved 129329 objects, 30644680 bytes, reason=1, used 24
            more bytes>
  <GC(64): stop threads time: 2306, start threads time: 2>
<AF[52]: managing allocation failure, action=3 (142678600/268368384)>
<AF[52]: managing allocation failure, action=4 (142678600/268368384)>
<AF[52]: clearing all remaining soft refs>
<GC(65): GC cycle started Wed Jul 30 14:50:28 2003
<GC(65): freed 12000 bytes, 53%% free (142690600/268368384), in 1717 ms>
   <GC(65): mark: 1581 ms, sweep: 136 ms, compact: 0 ms>
   <GC(65): refs: soft 8 (age >= 32), weak 0, final 0, phantom 0>
<GC(66): GC cycle started Wed Jul 30 14:50:32 2003
<GC(66): freed 1504 bytes, 53%% free (142692104/268368384), in 3724 ms>
   <GC(66): mark: 1563 ms, sweep: 606 ms, compact: 1555 ms>
   <GC(66): refs: soft 0 (age >= 32), weak 0, final 0, phantom 0>
   <GC(66): moved 6613 objects, 1296464 bytes, reason=1>
<AF[52]: managing allocation failure, action=6 (142692104/268368384)>
<AF[52]: totally out of heap space>
<AF[52]: completed in 11989 ms>
```

Note that it took three GC actions to find space for a 12M object to be allocated, including about four seconds in GC and two seconds in stopping threads.

**TIP**

Any allocation > 1M should be reviewed in the application, as it can be difficult to find these very large contiguous spaces in memory. This resulted in almost 57 percent of the last 21 seconds having been spent in GC.

#### 23.5.1.2 Soft References

Some applications need to cache large amounts of data. This impacts the size of the heap and the behavior of garbage collection. In some scenarios, the data that needs to be cached are difficult to determine, and the application errs on the side of caching too much data. The best way to do this is to use the java.lang.ref.SoftReference class to reference data in the cache. This allows for data that are not used over a certain number of Garbage Collection cycles to be freed up automatically.

#### 23.5.1.3 Memory Leaks

The verboseGC output can be used as a first step in determining if a memory leak exists. As the Garbage Collection cycles proceed, watch for a pattern in the percent free value at the end of each cycle. This may vary from cycle to cycle, but if it is consistently getting lower, it is highly likely a result of a memory leak in the application.

**TIP**

If a memory leak is occurring in an application, there may be little choice but to recycle a server. It may be worthwhile, however, to wait for java.lang.OutOfMemory exceptions to be thrown. It may be that the JVM is avoiding doing a compression of the heap during Garbage Collection (this is an expensive component of garbage collection) until it is absolutely necessary. Thus, the free heap may go down for 10 or 20 consecutive Garbage Collections, then may go back up as the JVM decides to do a more complete Garbage Collection with compaction and possible elimination of soft references.

**TIP**

If a java.lang.OutOfMemory exception is thrown, take a z/OS SVCDUMP. The IBM support team has tools and can use the dump to analyze the contents of the heap. In addition, both the WSAM and Introscope products contain powerful leak detection capabilities.

**Heap Settings** Ideal heap settings vary by application and environment. A smaller heap may require more frequent Garbage Collection, and a larger heap may have a longer cycle of garbage collection.

**TIP**

The “rule of thumb” is to keep your heap on the small side so that the cycles are short. This allows for more servers to be brought up as well. This is, of course, a balancing act. This is discussed in chapter 22, WebSphere Performance Tuning. We recommend starting with a 256M heap and modifying as needed based on the behavior of your workload under stress.

### 23.5.2 Just In Time (JIT) Compiler

The Just In Time Compiler (JIT) is essential to high performance Java applications. Its primary function is to take frequently executed methods, and compile the byte-code into machine code, so that it runs much faster. In nearly all environments, the settings for the JIT should not be modified. The JVM does a great job determining which methods to compile when based on how many

times a method has been executed since the JVM was started. In the current JDK 1.31 on z/OS, a method gets jitted after 2,000 executions.

**GOTCHA**

One big problem with modifying the jit threshold is that rarely executed methods can get jitted (i.e., methods run just a few times at initialization of the J2EE server). Each jitted method takes space in memory that could be used for other purposes.

### 23.5.3 Java Tracing

WebSphere V5 for z/OS is the first version to allow dynamic Java tracing. This is fully configurable from the Administrative Console just as on the distributed platforms.

**GOTCHA**

Now that it is dynamic, Java traces can be collected on production systems. You must remember, though, that this does add overhead. So, it is best to turn it on for short periods of time and hope to re-create the problem while it is on.

**TIP**

Some of the java trace commands we find most helpful on z/OS

F <WasProc>,DISPLAY,TRACE,JAVA (Display current settings)

F <WasProc>,TRACEJAVA='*=all=enabled' (Turn on all java tracing)

F <WasProc>,TRACEJAVA='*=all=disabled' (Turn off all java tracing)

And for more specific traces (i.e., jdbc, orb, transaction handling, …), simply use the adminconsole to set the traces, and note what the javatracespec is. It can then be set dynamically with this TRACEJAVA='<javatracespec>' command.

### 23.5.4 Java Stack Traces

The JVM in the Servant operates with a configurable number of "worker threads," along with some number of threads used by the JVM and the container for housekeeping. When applications run slowly, it is often helpful to get a "snap-shot" of the current call stack on each of the worker threads. You can extract call-stack information from a console dump. IBM support has several tools for this purpose such as TBack and SvcDump, but it can be done manually as well. Listing 23.3 provides a set of instructions for the viewing thread call-stack information from the Servant region with ASIDx 002C.

**Listing 23.3** Instructions for taking a console dump and getting thread stack traces.

```
/dump comm=(WebSphereStackTrace)  (dump console command)
/xx,ASID=(2C),SDATA=(RGN,TRT,CSA,NUC,PSA,GRSQ,LPA,SQA,SUM)   (this
is a WTOR where xx is the message#, may need to be broken up)
After the dump completes, bring it into IPCS
IP SETDEF ASID(X'2C')
IP SUMM FORMAT ASID(x'2C')
IP VERBX LEDATA 'NTHREADS(*)'
```

This will result in a display of the call-stack of all threads. All worker thread stacks will contain the Entry SR_ExecutionRoutine. Look for patterns in the top of the worker thread stacks. IBM Support has some additional tools (tback and svcdump) for viewing the threads in the stack. At the time of this writing, there are still some issues with being able to create a Java dump with WebSphere and the current JVM (kill -3 <pid>). When this is resolved, there may be an ability to use Thread Analyzer (an IBM tool that is freely available to analyze Java dumps from WebSphere J2EE servers as discussed in Appendix B, WebSphere Tooling Reference) against z/OS Java dumps (and the dumps will be simpler and quicker to take).

**TIP**

When looking at the worker threads, focus on frequently occurring call-stacks, and look down from the top to find the first recognizable method. This may be difficult if the server has not been up long, as non-jitted methods names do not show up in a dump.

## 23.6 Administration and Monitoring

General administration of WebSphere V5 on z/OS is the same as on distributed platforms and is not the focus of this chapter, except where it impacts performance.

### 23.6.1 First, Read the Manual

The place to begin creating a WebSphere environment that performs well is the *Operations and Administration* manual at the following URL: *ftp://ftp.software.ibm.com/software/webserver/appserv/zos_os390/v5/bos5b1001.pdf*. Chapters 9 and 10 in this manual provide excellent detail in setting up the containers to interact efficiently with the various z/OS subsystems.

### 23.6.2 RMF and WLM Overview

Using RMF, you can get some great insight into how your container and application are performing, and RMF generally has very low overhead. If WLM (which was covered in an earlier chapter) is configured with reporting and service classes, then RMF can provide an accurate and timely view of how your servers and workloads are performing. This may lead to refinement of the WLM policy. The *Operations and Administration* manual discusses many of the WLM details, and they are summarized from a WebSphere perspective in chapter 19 of this book. We are summarizing them here along with some special tips and examples.

### 23.6.3 WebSphere and WLM Classification

#### 23.6.3.1 Basic WLM Considerations

- Dynamic application environments simplify the WLM configuration such that you do not need to create new Application Environments for each server created.

- Be sure that System address spaces (Controller, Naming daemon, and Node Agent) run with a high priority (i.e., SYSSTC). These are generally short-running tasks, and they should have a high velocity goal.
- Servant address spaces need to be prioritized highly as well, although slightly lower than the Controller. These handle dispatching and nonenclave work (Garbage Collection, user threads, timers, etc.). Garbage collection is CPU-intensive, but, since it often stops all threads, a delay can impact all work in the server. User-created threads in the JVM (not something that is recommended, but it is sometimes necessary) will also run under the priority of the address space.
- Application environments for work running under the Servants (discussed later in this chapter) should be prioritized with a response-time goal. Keep in mind that the goal should reflect all work. If this application environment is serving static Web content as well as complex requests, the static content could skew the numbers.
- Classifying enclave and system work allows for granular RMF reporting to identify the actual "cost" of the various workloads.

**TIP**

It is best to serve static content out of a Web server instead of the Application Server. This avoids overhead for statics and allows the Application Server to use its cycles on business critical application work.

#### 23.6.3.2 Classifying Work into Enclaves

All of the business logic in the application occurs in the enclave created for the work. Enclaves are discussed in the IBM Redbook *OS/390 Workload Manager Implementation and Exploitation* which can be found at this URL: *http://www.redbooks.ibm.com/redbooks/pdfs/sg245326.pdf*.

The application enclave runs under the goal associated with its service class. By defining WebSphere work into meaningful service classes, proper goals can be set both for monitoring and for allowing z/OS to focus resources as needed to meet defined goals. The service class for an enclave is generally set in one of two ways:

- Default service class for the WLM subsystem. CB is a predefined subsystem in WLM (which has its roots in the product called Component Broker). You can define a default service class for the CB subsystem. If all WebSphere requests can run under one service class, this is the simplest and most straightforward way to define it
- Definition of a transaction class file in WebSphere to route URLs to transaction classes, then classification rules in WLM to use the WebSphere transaction class and classify the work into the appropriate service class. This is discussed in the next section.

**Routing WebSphere URLs to Specific Service Classes** Transaction Class Mapping allows for a mapping of URLs in a Servant to specific transaction classes. These transaction classes can then be mapped with WLM classification rules to WLM Service Classes and Reporting Classes.

Performance data for these WLM entities will then be reported in the RMF Workload Activity reports. The Mapping is done with a text file that you specify for the server via the admin console in ***Web Container _ Advanced Settings***. The syntax of the file is

```
TranClassMap <Server:Port> <url> <TranClass>
```

For example:

```
TranClassMap    *:*   /trade/*   TRADTCLS
```

This means that all work for the Trade application coming in to this server (or any server that uses this file since the server is wild-carded) will have a WLM transaction class of TRADTCLS. This can then be used in a WLM classification rule to route Trade work to a specific Service Class and Reporting class. An example of doing this is shown in Figure 23.3.

Note that this classification rule is within the *CB* subsystem, and it is type *TC* (transaction class). This will cause the work to run under the TRADSCLS service class, and RMF will report on it in the TRADRPT reporting class.

The RMF section later in this chapter will provide a sample of the report output and how this can then be used to determine information about the performance of the application in production.

### 23.6.3.3 Classifying Nonenclave Work

To get a full picture of how WebSphere is running on the system, it is recommended that you also isolate the performance of the Controller and Servant Address Spaces. The total cost of a WebSphere workload includes its portion of time in the Controller and Servant Address spaces, added to the time in the enclave (and while DB2 time is included in the enclave, CICS Transaction Gateway, CICS, IMS, or any back-end systems may also need to be factored in the equation). Figure 23.4 shows an example of the WLM classification rules within the STC subsystem to classify address spaces with the pattern MCV5S%% into a special Controller service class and to

```
  Subsystem-Type  Xref  Notes  Options  Help
 ------------------------------------------------------------------------------
                    Modify Rules for the Subsystem Type       Row 1 to 2 of 2
 Command ===> ____________________________________________ SCROLL ===> PAGE

 Subsystem Type . : CB          Fold qualifier names?   Y  (Y or N)
 Description  . . . cb_____________________________

 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
                                              DEFAULTS: CBFAST      ________
  ____  1  TC         MCTRDTRX ___                    MCCLUST1    ________
  ____  1  TC         TRADTCLS ___                    TRADSCLS    TRADRPT_
****************************** BOTTOM OF DATA ********************************
```

**Figure 23.3** WLM Classification rule to classify WebSphere URLs.

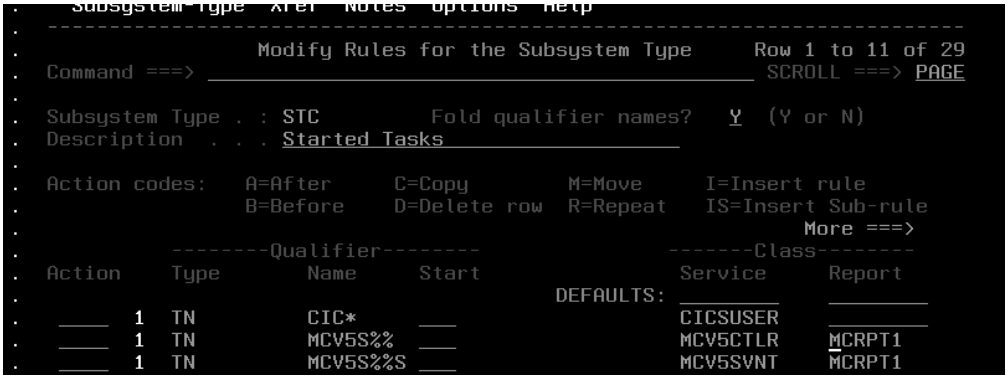


**Figure 23.4** WLM Classification rules to classify WebSphere Nonenclave work.

route MCV5S%%S into a Servant service class. These, along with the enclaves that will be tied to these address spaces, will report into the MCRPT1 reporting class. By having Controller, Servant, and enclave time reporting to the same reporting class, RMF does most of the work for us.

**Handling the Servant Address Spaces and Threads** In WebSphere, a server is defined as one Controller and one or more Servants. This section discusses how to control the number of Servant address spaces and the number of threads per address space. In general, it is best to let WLM handle the number of address spaces, but there are controls in case they are needed. We would suggest first *not* setting these and allow WLM to manage it. Changing these settings can have unforeseen effects on system resources and on Servers that process requests for multiple service classes.

Figure 23.5 shows an Administrative Console window under **Server _ Server Instance** with a check-box for handling of multiple Servants. If checked, then a minimum and maximum number of Servant address spaces can be selected. There are numerous factors to be considered in determining if and how to use this functionality, including:

- Is there a problem with how WLM is managing it?
- How many threads are in each Servant (covered in the next section)?
- How many service classes are being serviced by this server (refer back to TranClass Mapping)? A Servant address space can serve one service class at a time (due to storage isolation issues that are beyond the scope of this book). If work arrives for more service classes than are defined with the maximum setting, then a serious queuing problem could occur.

***BBOO_WORKLOAD_PROFILE*** is a setting for a server within **ORB SERVICE _ Advanced Settings**, used to determine the number of actual worker threads that will be started in each Servant. You should avoid assuming that more threads are always better. While this can

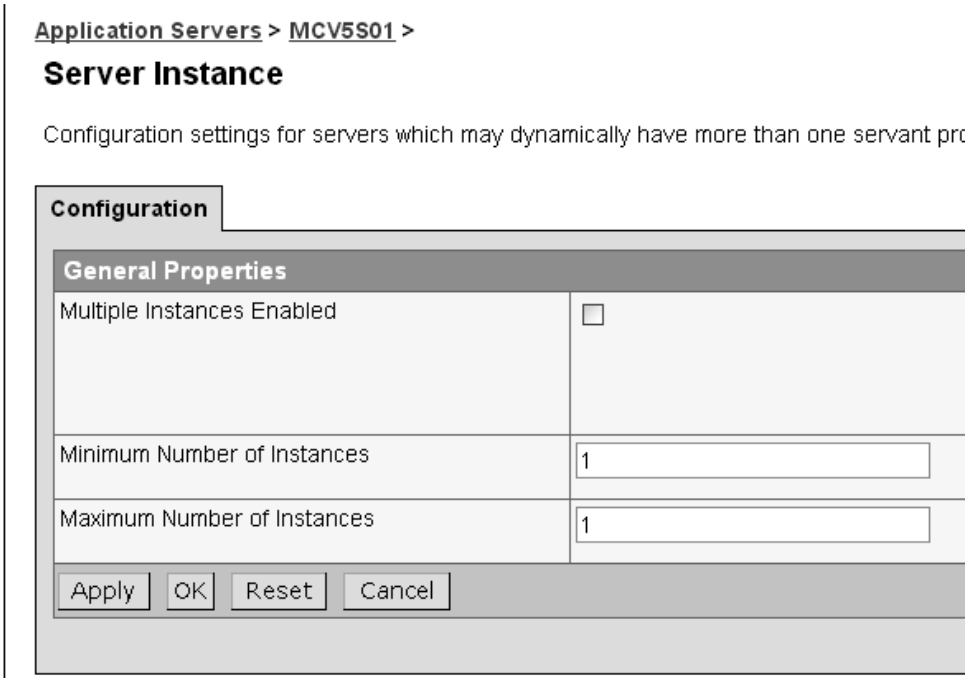


**Figure 23.5** Server Instance control over Minimum and Maximum Servants.

often be true in scenarios where an application makes calls to external servers, applications that are CPU-intensive within the Servant address space will not benefit from additional threads and may actually see a slight degradation in performance if too many worker threads exist in the address space (and remember that, unless instructed otherwise, WLM can create another servant address space if more threads are needed). The settings and the resulting number of worker threads are

**Table 23.2** WLM Settings and Worker Threads

| Setting | Worker Threads |
|---|---|
| Normal | Either 1 or 3 depending on isolation level |
| CPUBOUND | max((num_of_CPU-1),3) |
| IOBOUND | min(30,max(5,num_of_CPU * 3)) |
| LONGWAIT | 40 |

**TIP**

Keep your z/OS LPAR size and your application behavior in mind when deciding on how many threads or Servants to use. Threads are less expensive relative to system resources, but a well-tuned server can use six threads to keep four or five engines very busy. If the server makes calls to external servers (Soap, CICS, etc.), then additional threads may be helpful since threads can be in a wait state.

#### 23.6.3.4 RMF Considerations

Once your work is properly classified by WLM, RMF can provide a wealth of information. While anything in RMF could be relevant to performance, the initial areas to focus on are discussed below. Some important RMF/SMF considerations are

- Areas that highlight basic system performance (DASD response times, enqueues, the Coupling Facilities, Crypto hardware, HFS performance, …) should be reviewed regularly by your z/OS System Performance personnel.
- The RMF interval should be shorter during performance testing so that performance can be seen at a more granular level (less smoothing of the spikes). This is a dynamic change, and it can be undone dynamically.
- Workload Analysis can provide a great overall view of the application response time, application path-length, and resource consumption.
- Partition data, CPU information, and memory information provide a great deal of information about the overall system and how it is running.

The manual *Resource Management Facility Report Analysis (http://publibz.boulder.ibm.com/epubs/pdf/erbzra23.pdf)* provides an excellent resource for understanding the detail of all of the reports described here. You should review one set of reports for a time period of low WebSphere workload volume and one set of reports for a time period of high WebSphere workload volume. Collect the RMF data beginning just after the heavy workload starts and ending just before the heavy workload ends (avoid ramp up or initialization periods unless those are of specific interest). Like many other z/OS components, RMF creates SMF records within a preassigned range. The SMF records of most interest to WebSphere applications are

- RMF records—record types 70–79
- System Logger—record type 88
- USS/HFS—record type 92
- HTTP—record type 103
- DB2—record types 100–102

- CICS—record type 110
- MQ—record types 115–116
- TCP—record types 118–119
- WebSphere—record type 120

### 23.6.3.5 RMF Reports

There are many different RMF reports. We are going to focus on those that report most directly on WebSphere related information. Most examples are excerpts as the reports tend to be wider than can be nicely viewed in this format.

**Summary Report** Begin by checking here since the summary report gives an overview of how the overall z/OS system is performing. More detailed and specific reports need only be reviewed if there is an indication of trouble in this report. Listing 23.4 is an excerpt of a typical RMF Summary Report.

**Listing 23.4** RMF Summary Report.

```
1            R M F   S U M M A R Y   R E P O R T
                                                        PAGE  001
    z/OS  V1R3  SYSTEM ID XXX1    START 06/16/2003-09.29.00  INTERVAL 00.14.59
                RPT VERSION V1R2 RMF   END   06/16/2003-10.29.00 CYCLE 1.000
SECONDS
0
 NUMBER OF INTERVALS 4             TOTAL LENGTH OF INTERVALS 00.59.58
-DATE   TIME     INT   CPU   DASD  DASD  OMVS  OMVS SWAP DEMAND
 MM/DD HH.MM.SS MM.SS  BUSY  RESP  RATE   MAX   AVE RATE PAGING
006/16 09.29.00 15.00  47.4   4.6  1299     1     1 0.00   0.11
 06/16 09.44.00 14.59  44.6   4.1  1079     1     1 0.00   0.00
 06/16 09.59.00 14.59  41.6   4.8 976.4     1     1 0.00   0.00
 06/16 10.14.00 15.00  38.6   5.1 668.3     2     1 0.00   0.09
-TOTAL/AVERAGE         43.0   4.6  1006     2     1 0.00   0.05
```

This displays the obvious Date, Time, z/OS release, and Interval information. The important information is listed for each interval. The specific items to look for in this report are explained in Table 23.3.

**Table 23.3** Key Data from RMF Summary Report

| Term | Definition |
|---|---|
| Interval | Length of the interval for RMF reporting. This should be reduced when performance testing occurs to show a more granular report. |
| CPU Busy | CPU utilization during the interval (since engines are either fully utilized by this LPAR or NOT utilized by this LPAR, it is actually the percent of time that the engines were fully utilized by this LPAR). This number is relative to the processors allocated to this LPAR and is impacted by the weighting value of the LPAR. |
| Swap Rate | Number of swaps per second during this interval. If this is < 2%, then the Paging Report need not be reviewed. |
| Demand Paging | Page Fault Rate. Ideally, this will be 0. If it gets above 1 or 2, then adding memory can have a very positive effect. The old song says, "There's no paging like no paging." |
| DASD Resp | Average milliseconds required to complete an IO request. This should be < 10 milliseconds. If not, check the DASD reports to see if a volume may have caching enabled or if the spread of data is an issue. One exception is the volume containing the JES2 Checkpoint Data Set which tends to have higher utilization and longer response times than other DASD volumes. |
| DASD Rate | Tied to DASD Response. This is DASD activity per second across all devices. The higher this number, the more important the DASD Response. |

**Partition Data Report** This report shows a great deal of information about the entire system. It puts CPU utilization in context with image weights, and it shows the overall utilization of all LPARs on the image. Comparison of each LPAR with its associated weight helps determine which enclaves are consuming the most resources. Listing 23.5 is an excerpt from an example (with some line wrapping):

**Listing 23.5** RMF Partition Data Report.

```
1          P A R T I T I O N  D A T A  R E P O R T
                        PAGE    2
  z/OS V1R3      SYSTEM ID MVS1    DATE 06/16/2003            INTERVAL 15.00.715
                 RPT VERSION V1R2 RMF      TIME 09.29.00      CYCLE 1.000 SECONDS
```

```
-
 MVS PARTITION NAME                      MVS1
 IMAGE CAPACITY                           392
 NUMBER OF CONFIGURED PARTITIONS            4
 NUMBER OF PHYSICAL PROCESSORS             10
                    CP                     10
                    ICF                     0
 WAIT COMPLETION                           NO
 DISPATCH INTERVAL                    DYNAMIC
-
 ------- PARTITION DATA -------   — AVERAGE PROCESSOR UTILIZATION PERCENTAGES —
0              -CAPPING-  PROCESSOR-    LOGICAL PROCESSORS  -- PHYSICAL PROCESSORS --
NAME  S WGT DEF   WLM%  NUM   TYPE   EFFECTIVE    TOTAL  LPAR MGMT  EFFECTIVE  TOTAL
LPAR1  A 40  NO     0.0    6   CP       71.36    71.55       0.11      42.82  42.93
LPAR2  A  3  NO     0.0    2   CP        0.00     0.00       0.00       0.00   0.00
LPAR3  A 30  NO     0.0    5   CP       65.16    65.23       0.03      32.58  32.61
MVS1   A 27  NO     0.0    5   CP       47.05    47.37       0.16      23.53  23.68
*PHYSICAL*                                                  0.24               0.24
                                                           -----      -----  -----
   TOTAL                                                    0.55      98.93  99.47
```

Key Data from this Report (Listing 23.5) is highlighted in Table 23.4.

**TIP**

The highlighted sections in the report show that LPAR 1 is weighted at 40 (which in this case means 40 percent based on a total weight of 100). Note that it is actually consuming 42.93 percent of the entire z/OS machine during this interval. z/OS can balance this but, if this is a regular occurrence, then you likely have a CPU shortage here. If this is largely caused by WebSphere (which we can determine by the Workload report discussed later in this section), then either the workload volume is growing, or the application is not performing as well as when it was sized.

**Table 23.4** Key Data from RMF Partition Data Report

| Term | Definition |
|---|---|
| System ID | z/OS Image that this report was run on. |
| Number of Physical Processors | CPUs enabled on the system. |
| CP | In those processors, how many are enabled as general purpose processors. |
| ICF | This is for engines dedicated for Integrated Coupling Facility CPUs, and IFLs (Integrated Facilities for Linux) will also come under this category. |
| | **The remainder are for each LPAR.** |
| S | Status A=Activated; D=De-activated. |
| WGT | Weight of the LPAR. This is critical as it determines how much resource the z/OS image in this LPAR actually has. The number should be divided by the total weight (for all nondedicated LPARs) to determine the relative weight. If weight is DED, then this processor is dedicated to this engine (this is rare). Note: LPAR weights are relative. In order to interpret the weight of this LPAR, it should be compared to weights of the other LPARs on this processor. |
| Capping | This should always be **NO**, or one of the benefits of z/OS (sharing resources) will be compromised. Capping is sometimes used to protect a production LPAR, but it can cause some LPARs to be CPU-starved while cycles go un-utilized. You can protect an LPAR simply by giving it adequate weight. |
| Processor Num | Number of processors assigned to LPAR. This will impact CPU utilization reporting and potentially allow for greater concurrency. Total of Processor Num should be <= 3 * CP (from heading). That is, 3:1 is maximum good ratio for Logical Processors vs. Physical processors. |
| Avg Util Logical Processors Total | This should equal LPAR Busy Time Percentage from Summary Report. It includes LPAR overhead and is a percentage of time that the processors allocated to this LPAR were doing work for this LPAR. This is affected by many of the other columns as well. |
| Avg Util Physical Processors Total | This is the utilization spread out over all engines in the LPAR. This should be compared to the weight of the LPAR. If it is at or above the weight, then the LPAR was CPU-constrained during this interval. If performance is sub-par and this is < weight, then there is some other bottleneck. |

**CPU Activity** As shown in Listing 23.6 this provides CPU model information which factors into available MIPS. It also shows the CPU utilization information for this LPAR.

**Listing 23.6** RMF CPU Activity report.

```
1                                C P U  A C T I V I T Y        PAGE    1
  z/OS V1R3          SYSTEM ID XXX1     DATE 06/16/2003    INTERVAL 15.00.715
           RPT VERSION V1R2 RMF       TIME 09.29.00       CYCLE 1.000 SECONDS
-CPU  2064   MODEL  210
0CPU   ONLINE TIME LPAR BUSY  MVS BUSY  CPU SERIAL  I/O TOTAL  %I/O INTERRUPTS
NUMBER PERCENTAGE  TIME PERC  TIME PERC NUMBER  INTERRUPT RATE HANDLED VIA TPI
 0     100.00      53.20      67.44     049823    694.2      3.08
 1     100.00      50.54      64.23     149823    645.9      2.83
 2     100.00      46.69      60.27     249823    560.3      2.57
 3     100.00      43.77      57.01     349823    564.2      3.26
 4     100.00      42.65      55.76     449823    651.2      5.01
 TOTAL/AVERAGE     47.37      60.94                3116      3.37

 SYSTEM ADDRESS SPACE ANALYSIS         SAMPLES =    900
-       NUMBER OF ASIDS                DISTRIBUTION OF QUEUE LENGTHS  (%)
TYPE  MIN    MAX     AVG     0    1    2    3    4    5    6  7-8  9-10  11-12
--- ----- ----- -------- ---- ---- ---- ---- ---- ---- --- ---- ----  ----
IN
READY  1     29      5.3    0.0  8.4  13.6 17.1 15.2 10.4 7.7  9.0 6.5    5.0
```

Key Data from this Report (Listing 23.6) is highlighted in Table 23.5.

**Table 23.5** Key Data from RMF CPU Activity Report

| Term | Definition |
|---|---|
| z/OS V1R3 | OS specification. |
| System ID XXX1 | Name of the LPAR that collected the RMF records. |
| -CPU 2064 Model 210 | Processor specification, helps determine the overall capacity of the system, used in conjunction with the *Partition Data Report*. |
| LPAR Busy Time Perc | Percent of time this CPU was dispatched with work from this LPAR during this interval In LPAR mode (the norm), this number is more important than MVS Busy time. |
| MVS Busy Time Perc | Percent of online time this CPU was busy overall during this interval. |
| IN READY queue | Indication of queuing for CPU. Depending on number of processors. Greater than 90% of samples should show with queue length less than 3 * number of processors (this is the case here). |

**Workload Activity** All of the preceding reports focused on the details and health of the overall system. If the recommendations from the WLM section were taken, and the definitions made to measure the WebSphere workload in a granular manner, then the Workload Activity reports on Service and Report Classes can provide a great deal of info on the actual work flowing through WebSphere. The following excerpts from Workload activity reports were taken from true runs of a WebSphere Application. We show the Controller Service Class (MC5CTLR), the Servant Service Class (MC5SVNT), and the enclaves (MCCLUST1).

Listing 23.7 is for the enclave as it contains some information not found (or not valuable) in the other Workload Activity Reports.

**Listing 23.7** Workload Activity report, enclave service class.

```
REPORT BY: POLICY=WEB_RAL    WORKLOAD=CB          SERVICE CLASS=MCCLUST1
                                                  CRITICAL     =NONE

TRANSACTIONS    TRANS.-TIME  HHH.MM.SS.TTT   --SERVICE--   -SERVICE RATES-
AVG       6.00  ACTUAL                 65   IOC          0    ABSRPTN      8950
MPL       6.00  EXECUTION              40   CPU      13041K   TRX SERV     8950
ENDED    36198  QUEUED                 25   MSO          0    TCB          93.4
END/S   148.99  R/S AFFINITY            0   SRB          0    SRB           0.0
#SWAPS       0  INELIGIBLE              0   TOT      13041K   RCT           0.0
EXCTD        0  CONVERSION              0   /SEC     53674    IIT           0.0
AVG ENC   6.00  STD DEV                197                    HST           0.0
REM ENC   0.00                                                APPL %       38.4
MS ENC    0.00

   --RESPONSE TIME---  EX    PERF   AVG    --USING%--  ---- EXECUTION DELAYS %
        HH.MM.SS.TTT    VEL  INDX  ADRSP   CPU   I/O  TOTAL  QMPL  CPU
GOAL    00.00.02.000  90.0%
ACTUALS
3090            99.0%  0.0%  0.5    9.9   0.0   0.0   52.1  45.1  7.0

                        --------RESPONSE TIME DISTRIBUTION--------
   ----TIME----     --NUMBER OF TRANSACTIONS--     ------PERCENT------
   HH.MM.SS.TTT    CUM TOTAL        IN BUCKET     CUM TOTAL   IN BUCKET
<  00.00.01.000        35828            35828          99.0        99.0
<= 00.00.01.200        35840               12          99.0         0.0
<= 00.00.01.400        35842                2          99.0         0.0
<= 00.00.01.600        35842                0          99.0         0.0
<= 00.00.01.800        35842                0          99.0         0.0
<= 00.00.02.000        35842                0          99.0         0.0
<= 00.00.02.200        36196              354           100         1.0
```

```
<= 00.00.02.400        36197              1          100          0.0
<= 00.00.02.600        36198              1          100          0.0
<= 00.00.02.800        36198              0          100          0.0
<= 00.00.03.000        36198              0          100          0.0
<= 00.00.04.000        36198              0          100          0.0
<= 00.00.08.000        36198              0          100          0.0
>  00.00.08.000        36198              0          100          0.0
```

Key Data from this Report (Listing 23.7) is highlighted in Table 23.6.

**Table 23.6** Key Data from RMF Workload Activity Report

| Term | Definition |
|---|---|
| Workload CB | Workload classification from WLM. All WebSphere work is in the CB subsystem. |
| Service Class MCCLUST1 | WLM service class in which this workload runs. This is the class for the enclave work. |
| Transactions AVG 6.00 | Number of transactions concurrently being processed (on average). |
| Transactions END/S 148.99 | Number of transactions in this class ended per second on average for this interval. |
| TransTime Actual 65 | Average response time during this interval (in milliseconds). |
| Service Rates APPL % 38.4 | Average % of a single CPU used by running transactions. Note multi-processor systems can have a number > 100% here as the workload may consume cycles on any processor in the LPAR. |
| Goal 2 seconds 90% | WLM response time goal for this service class. |
| Response Time Distribution | Percentage distribution of response times (full report includes histogram). Breakouts are based on goal. |

You can get a rough feel for CPU Time per transaction within the enclave if you know the number of MIPS per engine of your processor (could look this up based on 2064/210 from the CPU Activity report). The formula is: APPL % * MIPSPerEngine / (EndSecond * 100). In this example, this would be 38.4 * 230.75 / (148.99 * 100) = .59 MIPS/tran. At this point, this does not include CPU in the Controller or Servant (which is usually light) or the time spent in CTG or CICS (DB2 time is reported in the enclave, but CTG and CICS are not). This example is for a simple transaction driving an EJB; real work with DB2, XML, and so on, will likely be notably longer.

**GOTCHA**

It is not unusual for WebSphere Servers to do re-directions to other URLs. If this URL is handled by this same server and transaction class, then the END/S number may be double what the client perceives. Also, some Web pages may involve multiple individual requests, and again the perception of the client may be different than the transactions as RMF reports them.

Listing 23.8 shows a similar report only for the Controller.

**Listing 23.8** Controller Address Space Service Class report.

```
REPORT BY: POLICY=WEB_RAL    WORKLOAD=CB         SERVICE CLASS=MC5CTLR
                                                 CRITICAL     =NONE
                             DESCRIPTION  =MikeC WASV5 Controller ASIDs

TRANSACTIONS   TRANS.-TIME HHH.MM.SS.TTT  --SERVICE--  -SERVICE RATES-
AVG      3.00  ACTUAL                 0  IOC      4875   ABSRPTN      3885
MPL      3.00  EXECUTION              0  CPU     15030   TRX SERV     3885
ENDED       0  QUEUED                 0  MSO      2812K  TCB           0.1
END/S    0.00  R/S AFFINITY           0  SRB        16   SRB           0.0
#SWAPS      0  INELIGIBLE             0  TOT      2832K  RCT           0.0
EXCTD       0  CONVERSION             0  /SEC    11655   IIT           0.0
AVG ENC  0.00  STD DEV                0                  HST           0.0
REM ENC  0.00                                            APPL %        0.0
MS ENC   0.00
```

As is normal here, the Controller took a very small amount of CPU, and there is no workload distribution since there are no measurable transactions in this address space. The Servant address space is similar (and, thus, not shown). If there is a significant amount of CPU consumed in the Controller or Servant, it is worth further research to determine why. Typical Servant problems could be high garbage collection processing, user threads doing application work in the region, or monitoring tools that do much of their work in the Servant and not in the enclaves.

The CTG and CICS service classes are also similar, but they can consume much larger amounts of CPU. Unless the workload is highly isolated relative to other CICS workload, it may be difficult to isolate the amount of CTG and CICS work that this application does relative to the overall CICS workload. For this purpose, these service classes were not included in the reporting class. It is always worth reviewing these service classes, however, to get an idea for resources being consumed in the various elements on the system.

### 23.6.4 DB2

DB2 is likely the most common back-end or data store employed by WebSphere on z/OS. This will focus on use of the standard Type 2 Java Database Connectivity (JDBC) driver. We focus on monitoring performance from a JDBC and DB2 perspective. Tuning DB2 is beyond the scope of this chapter.

#### 23.6.4.1 JDBC Tracing and Reduction

Java Data Base Connectivity (JDBC) is the way that WebSphere or any Java application can connect to DB2. Most current JDBC drivers implement the JDBC 2.0 spec. Soon most will be going to the JDBC 3.0 spec. There are four types of JDBC drivers based on how they communicate between the database client and the database server. The types are

Type 1: JDBC/ODBC bridge. Java interfaces to an ODBC layer in order to get to the database server.

Type 2: Full Java provider that accesses a local database.

Type 3: Full Java provider that interfaces (usually over a network) to a specialized daemon or code on the database server machine which then communicates with the database server.

Type 4: Full Java provider that communicates directly to the database server.

Note that Types 2 and 4 are the only commonly used types. Type 2 can be used to a local database client, which then uses database mechanisms (i.e., DRDA) to access a remote database server.

In addition to JDBC specification level, and driver type, with DB2, there is now a migration on z/OS from the legacy Type 2 driver, to the universal driver that can operate as a Type 2 driver or a Type 4 driver. JDBC trace can be set through the WebSphere Administrative Console on the universal driver. Since most setups prior to DB2 Version 8 and WebSphere 5.02 (W502002), this JDBC trace will focus on the legacy driver on z/OS which is Type 2.

JDBC Tracing can impact performance by more than 50 percent, and it is nondynamic (i.e., turning it on and off requires a recycle of the server), so it should only be enabled for verification and/or for problem determination. JDBC Trace information is very valuable in understanding what is flowing between the application and DB2. To enable JDBC trace, modify the *db2sqljJDBC.properties* file (specified in *was.env* in your WebSphere configuration directory for this server) and set the lines as shown in Listing 23.9.

**Listing 23.9** db2sqljJDBC.properties directives to enable a trace.

```
DB2SQLJ_TRACE_FILENAME=</yourdir/yourhfsfile>
DB2SQLJ_TRACE_BUFFERSIZE=1024
```

Normally, the *DB2SQLJ_TRACE_FILENAME* directive should be commented out or removed as this turns off the trace.

This trace will create two files: */yourdir/yourhfsfile* and */yourdir/yourhfsfile.jtrace*. The jtrace (Java trace) is very large and very useful. It can be used to truly understand the flow of requests and replies between the application and DB2. A typical flow, for example, of getting a connection would appear as shown in Listing 23.10.

**Listing 23.10** JDBC Jtrace excerpt from a prepared statement.

```
<2003.06.19 23:53:27.692> <Entry>   <prepareStatement>
<COM.ibm.db2os390.sqlj.JDBC.DB2SQLJConnection@720d0865>
<WebSphere t=009c1288>
```

```
    -- <p#1=SQLText=SELECT distinct CORP1_STATE_CD, STATE_CODE
FROM DFBP50P.tst_bus_region_typ WHERE EFF_DT <= CURRENT DATE  AND
(EXP_DT > CURRENT DATE OR EXP_DT IS NULL)>

<2003.06.19 23:53:27.696> <Entry>   <Initialize>
<COM.ibm.db2os390.sqlj.JDBC.DB2SQLJJDBCCursor@218ec865>
<WebSphere t=009c1288>
    --
<p#1=this=COM.ibm.db2os390.sqlj.JDBC.DB2SQLJJDBCCursor@218ec865[pS
TMT=0]>
    --
<p#2=conn=COM.ibm.db2os390.sqlj.JDBC.DB2SQLJConnection@720d0865[pC
ONN=2938eb00]>
    -- <p#3=Type=3>
    -- <p#4=role=-101>
    -- <p#5=nativeSQLText=SELECT distinct CORP1_STATE_CD,
STATE_CODE FROM DFBP50P.tst_bus_region_typ WHERE EFF_DT <= CURRENT
DATE  AND (EXP_DT > CURRENT DATE OR EXP_DT IS NULL)>
    -- <p#6=DB2StmtType=0>
    -- <p#7=numParameters=0>
    -- <p#8=metaData=0>
    -- <p#9=ResultSetCount=0>
    --
<p#10=JDBCProfile=COM.ibm.db2os390.sqlj.JDBC.DB2SQLJJDBCProfile@500c
c865>
    -- <p#11=Section=350>
    -- <p#12=cursorName=DB2OS390HOLD100>

 <2003.06.19 23:53:27.698> <Exit>    <Initialize>
<COM.ibm.db2os390.sqlj.JDBC.DB2SQLJJDBCCursor@218ec865>
<WebSphere t=009c1288>

 <2003.06.19 23:53:27.698> <Entry>   <prepare>
<COM.ibm.db2os390.sqlj.JDBC.DB2SQLJJDBCCursor@218ec865>
<WebSphere t=009c1288>
    --
<p#1=COM.ibm.db2os390.sqlj.JDBC.DB2SQLJJDBCCursor@218ec865[pSTMT=5
48743dc]>
    --
<p#1=DB2SQLJConnection=COM.ibm.db2os390.sqlj.JDBC.DB2SQLJConnectio
n@720d0865[pCONN=2938eb00]>
```

```
    -- <p#2=TransactionState=1>
    --
<p#3=COM.ibm.db2os390.sqlj.JDBC.DB2SQLJConnection@720d0865[pCONN=2
938eb00]>

 <2003.06.19 23:53:27.756> <Exit>    <prepare>
<COM.ibm.db2os390.sqlj.JDBC.DB2SQLJJDBCCursor@218ec865>
<WebSphere t=009c1288>
```

JTrace contains a wealth of information if you take a bit of time to understand the format and learn to extract what you need. The non-JTrace file contains information that is likely more valuable to a DB2 expert than to an application developer or WebSphere administrator. It is a binary file that can be formatted to be viewable with the commands shown in Listing 23.11.

**Listing 23.11** JDBC formatted and flowed traces.

```
db2sqljtrace fmt fname >fname.fmt  (to get formatted version)
db2sqljtrace flw fname >fname.flw  (to get flowed version)
```

Several configuration and application problems can be diagnosed with the jtrace files, including:

- Connections not being closed and returned to the pool.
- More SQL calls being executed than expected.
- Incorrect SQL being generated by an application.
- Commit/rollback issues (conn.setAutoCommit(true) issue).
- Improper usage of prepared statements (preparing statements that do not get executed). Prepared statements are good to use as they can result in more hits on the dynamic cache.

All of these can manifest themselves as performance problems.

### 23.6.5 DB2 Tracing

DB2 Tracing on z/OS is completely dynamic and is excellent for understanding the behavior of your application, relative to DB2, at a high level. In addition to being dynamic, most DB2 systems are already running the type of tracing that can give an excellent high level view of the DB2 subsystem.

**GOTCHA**

If you use the legacy JDBC provider for z/OS and you have a local DB2 that passes the requests through to a remote DB2, then your analysis should be of the remote DB2. Specifically, if the local DB2 is trusted on the remote DB2, the signons on the remote DB2 should be low compared with the number of requests.

A common trace string is shown in Listing 23.12.

**Listing 23.12** Standard DB2 trace options.

```
-<Db2CmdPref> start trace(stat) class(1 3 4 5 6) dest(smf)
-<Db2CmdPref> start trace(acctg) class(1 2 3) dest(smf)
```

These traces are generally already running on most DB2 systems. Report on these data by doing an SMF extract of type 100–102 records for the time period in question. Then run a DB2/PM postprocessing job against the data specifying:

**Listing 23.13** DB2/PM control cards to review Accounting and Statistics Data.

```
DB2PM ACCOUNTING(REDUCE,
                 REPORT(LEVEL(SUMMARY,DETAIL),ORDER(PLAN)))
DB2PM STATISTICS(REDUCE,TRACE,
                      REPORT(LEVEL(SUMMARY,DETAIL)))
DB2PM EXEC
```

This will generate several reports.

#### 23.6.5.1 System Parameter Report

The first item we review is the DB2PRMDD output. This provides all of the information from the ZParms. ZParms are DB2 system parameter settings that your DB2 System Programmers can modify if needed (but it does require a DB2 recycle). The items to focus on are

- MAX NO OF USERS CONCURRENTLY RUNNING IN DB2 (CTHREAD). This should be at least 300 depending on the expected volume and concurrency.
- MAXIMUM KEPT DYNAMIC STATEMENTS (MAXKEEPD). This should be at least 500 as it controls the number of prepared statements kept in the cache (and, thus, do not need to be re-prepared).
- CACHE DYNAMIC SQL (CACHEDYN). This turns on the dynamic caching which is critical to performance in a dynamic SQL environment (i.e., JDBC).
- MAX NO OF BATCH CONNECTIONS (IDBACK). This may need to be increased with high concurrency WebSphere applications (we use at least 250).
- CHECKPOINT FREQUENCY (CHKFREQ) should be increased if possible to reduce DB2 overhead (we set this to 50,000)

#### 23.6.5.2 Statistics Reports

You should then do a quick review of the statistics report which provides a view of overall DB2 subsystem health. The items to review are

- Volume of requests in SQL DML and SQL DDL sections (focus on prepares and describes).
- IDENTIFY and SIGNON requests under SUBSYSTEM SERVICES. This will identify the amount of time DB2 spends in authenticating connections. This will be impacted by the RunAS settings on EJBs and security settings from servlets.
- Various locking information under LOCKING ACTIVITY.
- Major focus on prepare requests in DYNAMIC SQL STMT with a focus on FULL vs. SHORT (you want almost all short) as summarized by GLOBAL CACHE HIT RATIO (%).

### 23.6.5.3 Accounting Reports

The accounting report is similar to the statistics report, only here you can focus on a specific DB2 plan. If using DB2 local to WebSphere on z/OS, the plan should be DSNJDBC. It is also split by Primary Authid on the thread, which can make for a long report requiring a great deal of filtering effort. In some odd scenarios, with distributed usage, it is listed as ?RRSAF (work not actually occurring under a plan). For the appropriate plan, you will want to review the same type of data as you did in the Statistics Report, plus the following:

**Listing 23.14** DB2/PM accounting report information.

```
PRIMAUTH: SRNAME1  PLANNAME: DSNJDBC

 ELAPSED TIME DISTRIBUTION                          CLASS 2 TIME DISTRIBUTION
 -------------------------                          ---------------------- APPL
|==================================> 92%   CPU     |==> 4%
DB2    |===> 7%                                    NOTACC ==================> 88%
SUSP   |> 1%                                       SUSP   |====> 8%

AVERAGE     APPL(CL.1) DB2 (CL.2) IFI (CL.5)   CLASS 3 SUSPENSIONS  AVERAGE TIME
--------- --------- ----------- ---------   --------- -----------   ELAPSED TIME
1.212208  0.093436       N/P   LOCK/LATCH(DB2+IRLM)      0.000202
 NONNESTED    1.212208  0.093436       N/A   SYNCHRON. I/O             0.000000
 STORED PROC  0.000000  0.000000       N/A    DATABASE I/O             0.000000
 UDF          0.000000  0.000000       N/A    LOG WRITE I/O            0.000000
 TRIGGER      0.000000  0.000000       N/A   OTHER READ I/O            0.000000
Continuation Lines w/in report
               AV.EVENT   HIGHLIGHTS
               --------   --------------------------
                  0.36    #OCCURRENCES    :    74873
                  0.00    #ALLIEDS        :        0
                  0.00    #ALLIEDS DISTRIB:    74873
                  0.00    #DBATS          :        0
                  0.00    #DBATS DISTRIB. :        0
```

- Under ELAPSED TIME DISTRIBUTION look for Average ELAPSED TIME in Class 1 and Class 2 (Class 1 is total time from client connect to disconnect, Class 2 is time spent actually doing DB2 work). If Class 2 is a small percentage of Class 1, then DB2 is not likely a big factor.
- Review Class 3 (suspensions) to see if anything in particular stands out as having long delays.
- A review of the DSNJDBC2 vs DSNJDBC3 packages can highlight the isolation level of the work being done. DSNJDBC2 work is more efficient than DSNJDBC3 work (lower isolation level). It is rare to see significant work under the DSNJDBC1 or DSNJDBC4 package.
- Consider the NOTACC (not accounted for) time. If it is high (as it is in this example), it could be a sign of problems relative to swapping or other unaccounted-for time.

## 23.7 HTTP Front-End Handlers for WebSphere

Other chapters such as the Plug-in chapter covered the options and "how-to" portions of getting workload to the WebSphere servers. This section focuses on performance implications of the various techniques. The factors we look at specifically are listed here, and a summary table at the end of the section provides concise analysis.

- Impacts on performance, including usage of CPU resources
- Reliability
- Benefits/Functions
- Costs

While the overall options for front-ending have many permutations, including hardware, software, and so on, we are going to focus on just a few. Clearly, in some cases, these can be combined, and other options exist, but these are some of the most commonly used options. You will note that the older WebSphere V4 plug-in using RMI/IIOP is not considered as its performance is not as good as other options, and it is not supported in WebSphere V5.

### 23.7.1 Browser/Workload Direct to Controller HTTP/HTTPS Transport

The simplest setup is to have the source of the workload (usually browsers) pointed directly to the HTTP or HTTPS transport in the WebSphere Controller. This will likely provide the fastest solution, but with few options and functions available. In WebSphere V5, there will be no http logging performed (access, agent, referrer, …), although V5.1 does provide this (this is supported on distributed platforms, but not in the current HTTP Transport on z/OS ). For the standard dynamic

WebSphere workload, where everything is served out of WebSphere on z/OS, this uses minimal CPU time for routing. Functionally, however, it limits options relative to distributing the work and/or caching results. All requests are handled by WebSphere, which is not the low-cost solution for serving or caching static Web content. This is a reliable solution as there are fewer components involved. It is also a low-cost solution since no additional software is necessary.

### 23.7.2 z/OS IHS HTTP/HTTPS Plug-in Forwarding

This is a recent z/OS IBM HTTP Server (IHS) function modeled after the plug-in functionality on distributed Web servers. It has a minimal impact on performance since IHS is efficient, but it will use CPU resources in the front-end, so, if your system is under high utilization, it could compete with WebSphere for cycles. It is highly reliable and, for those who have historically used IHS, configuration is quite easy. If it is being used for several Server Instances, it will keep session affinity intact. It allows for all of the logging and debug features of IHS to be used in conjunction with those provided by WebSphere V5 . Since it is part of z/OS, IHS does not incur additional financial costs. This infrastructure allows for splitting workload (i.e., serving static content directly from IHS with or without caching). It is usually the case that serving static files from a Web server is faster than serving them out of an application server, and this will also avoid some TCP hops.

### 23.7.3 Distributed HTTP/HTTPS Plug-in Forwarding

This is a simple and efficient way to front-end the WebSphere z/OS HTTP transport. Weighted averages can be used to select z/OS and non-z/OS servers. Setting up the plugin-cfg.xml for a z/OS server is no different than setting it up for a distributed server and WebSphere V5 on z/OS can generate plug-in-cfg.xml files as well. It is important to be sure that there is enough available bandwidth on the machine running IHS to avoid creating a bottleneck in presenting workload to z/OS. As with the z/OS IHS, this allows for the workload to be split out as needed. Further information on the plug-in can be found in chapter 10 (The WebServer Plug-in) and the plugin-cfg.xml appendix.

### 23.7.4 WebSphere Edge Components

WebSphere Edge Components (formerly known as “The Edge Server”) provides a superset of the facilities that the distributed IHS plug-in provides. With WebSphere V5, the Edge Components can be exploited for all of their capabilities such as caching Web content, routing, and so on. Full discussion of the Edge Components is beyond the scope of this chapter, but the Edge Components can be used to reduce workload on back-end servers, reduce network latency and distance, improve response time to the browser, and much more. It can be especially valuable in scenarios where the back-end servers are highly utilized.

**Table 23.6** Summary Table of HTTP Front End Handlers

| | Performance | Reliability | Benefits | Costs |
|---|---|---|---|---|
| Browser Direct to Controller | Optimal for dynamic workload, inflexible in that all work is done out of WebSphere AppServer. Can provide lowest Response Times. | Highly Reliable, no additional points of failure introduced. | Low, little logging, no splitting up workload. | Low |
| z/OS IHS Plug-in | Very good, extra TCP hop, but ability to serve static requests out of IHS is a plus. Some extra CPU cost. | Highly Reliable. One extra feature, but IHS is stable. | Logging, static caching, splitting load. | Low |
| Distributed Plug-in | Very good, similar to z/OS IHS but does require extra z/OS CPU resources. Make sure IHS box has sufficient available bandwidth. | Highly reliable. Extra hop over network, but IHS highly reliable. | Same as for z/OS IHS but can place boxes closer to clients. | Low to Mid |
| Edge Server | Very good, including dynamic caching. | Highly reliable, many features to avoid single point of failure. | Can be complex setup, but dynamic caching features are worth it. | Mid |

### 23.7.5 Note About Sysplex Distributor

**Sysplex Distributor** can play a role in many of the scenarios mentioned. In using Dynamic Virtual IP Addressing (DVIPA) or Distributed DVIPA, it is simple to vary servers on and off with little or no impact to clients. In addition, this can be used as a workload distributor. Sysplex Distributor and Distributed DVIPA do not consume a great deal of resources.

Table 23.6 is a summary of HTTP front-end handlers.

## 23.8 Cookbook Approach to Problem Resolution

When performance problems occur on WebSphere V5 for z/OS, there are a myriad of places to look. The best solution we have found is to divide and conquer, and run the least intrusive tests first (steps 1 through 5). The more intrusive tests generally require a test system and a load generator.

### 23.8.1 Nonintrusive Procedures

These steps can be performed on a production system and do not involve modifying the code. They can be executed by either administrative or development personnel. An additional benefit to these is that they involve full workload (production or created stress). Thus, the behavior of the system will include all concurrency issues, as well as issues involved in an individual request.

#### Step 1: Look for the obvious

- Are there traces on (in WebSphere or on the system) that impact performance?
- Are the container configurations consistent with best practices?
- Are the system tuning items (covered in *Operations and Administration)* done?
- Are the performance goals reasonable, based on the application and the resources?
- Review VerboseGC and look for memory leaks?

#### Step 2: Simple controlled test

Run a simple, controlled (and easily repeatable) test and review the RMF data.

If response time or throughput goals are not being met and the CPU is *not* under heavy utilization, there are likely external delays. Look for GRS contention and other delays in the RMF reports. Look at the workload analysis for the enclaves to see what may be delaying them on the system. Look at the APPL percent versus the response time in high and low usage times.

If the CPU is heavily utilized, determine the CPU seconds /Tran (discussed earlier) versus the transaction volume. If the CPU Seconds/Tran are higher than expected, then application tuning may be in order (Java tracing, Jinsight, and other actions discussed in the remainder of this section).

#### Step 3: Dump

Taking one or more snapshots during a period of high volume can provide insight into where the threads are spending their time. Use the method of displaying the trace-backs discussed earlier. Focus on call-stacks that contain the SR_ExecutionRoutine CSECT. Look for patterns in the tops of the call-stacks, especially leading into monitor locks, and so forth. This can often detect bottlenecks in such areas as:

- Calls to back-end systems that are taking too long
- Calls to DB2
- Synchronized sections of an application or third-party product in use
- Excessive logging

#### Step 4: Container tracing

Starting in WebSphere V4 (and improving in V5), container tracing can be turned on and off dynamically. This is a great way to help find the delays in your workload. Lightweight traces can be turned on with the modify commands described in chapter 3 of *Operations and Administration*. The best details on the tracing can be found in Appendix A of *Assembling J2EE Applications*. A useful trace option is to turn on the basic trace with the console command:

```
/f <ServerProc>,tracebasic=(3,4,5,6)
```

Then create or allow some workload to occur, then:

```
/f <ServerProc>,traceinit
```

#### Step 5: Java Tracing

WebSphere V5 provides dynamic Java tracing which aids tremendously in isolating problems. Via the *Administrative console* or via the commands in chapter 3 of *Operations and Administration,* very detailed Java tracing can be turned on and off dynamically.

### 23.8.2 Intrusive Procedures

These tests involve modifying code or stopping and starting servers. Thus, they are best done on test or performance systems, and they generally do *not* involve running with stress on the system.

#### Step 6: WSAD profiling or Jinsight

Before the time of publishing, Jinsight functionality was replaced by WSAD profiling, and it is recommended that this be used instead.

Jinsight is an IBM tool that allows all method calls in a JVM to be captured and timed. Information on Jinsight (including download instructions) can be found at *http://www106.ibm.com/developerworks/java/library/j-jinsight/*.

WSAD V5 incorporates much of the functionality of Jinsight. By using the profiling features in WSAD, the developers should be able to get a feel for the expense of each of the items in the call.

Jinsight (which is not a supported product) has the additional benefit of running this profiling on the z/OS system which may have some different behaviors than the development system. Jinsight tracing can be done with the generally available Jinsight 2. In reviewing Jinsight output, we have found the following procedure to work best:

- Have an IBM person use Jinsight Live to capture multiple separate traces, one for each key request and/or data path.
- For each result, load the file into Jinsight by starting Jinsight, and selecting *File* → *Open a Trace File* and selecting the appropriate file. Then press the *Load* button.

- Select *Views* → *Execution* and you will be presented with the various work on each thread. Use the mouse to turn most of the work on one or more threads yellow (start near the left side of the work graphic, but not all the way to the left).
- Select *Selected* → *Drill down from selected items* → *Call Tree*. This provides an excellent view of the amount of time taken by each method of the call. Focus more on the percentages than on the actual contribution column.

**Step 7: Footprinting application**

If none of the steps already discussed have isolated the problem to the point where it can be fixed, then it is time to begin footprinting the code. This can be done elegantly with Log4J or JRAS, or in a more homegrown method with System.out.println() or something of the sort. This should help to isolate the methods that cause the problem.

## 23.9 Summary

You should now be familiar with the following concepts:

- The measurements of performance:
  - Response Times
  - Resource Utilization
  - System throughput
  - Scalability
- Understanding the importance of repeatablitiy and ability to re-create problems in assessing impacts of changes.
- Understanding the impact of tuning of z/OS subsystems to the overall performance of WebSphere on z/OS.
- Estimating the impact of various topologies on performance.
- Assessing the health of the JVM and garbage collection.
- Using the monitoring tools for WebSphere and z/OS and know which apply to different types of problems.
- Selecting the front-end topology that best meets your needs for performance, reliability, and simplicity.

CHAPTER 25

# Problem Prevention and Determination Methodology

## Objectives

This chapter covers the following concepts:

- Problem prevention best practices
- Change control best practices
- WebSphere best practices
- Working with IBM WebSphere support

## 25.1 Problem Prevention Best Practices

### 25.1.1 Testing Best Practices

One of the best ways to avoid and flush out problems prior to releasing a new application into the production environment is to properly test it. To most, the concept of testing is an obvious prerequisite to putting an application into the production environment. However, it is important to note that the key is not just "testing," but "proper testing". To paraphrase a famous statement, it is not whether you test, but *how* you test. A large percentage of the problems that occur in production environments can be prevented if the application is properly tested. It is the goal of this section to describe a test methodology that, when followed, will result in a significantly reduced risk of production outages.

#### 25.1.1.1 Properly Scaled Tests

Prior to testing an application, it is important to ensure the environment in which it is tested is appropriate for that application. An appropriate test environment for an application is one that is a scale model of its production environment, including all components from Web server to data-

base. Having an exact replica of the production environment for testing may not always be economically feasible; however, it is highly recommended that the test environment be an accurate model of the production environment.

**What Does It Mean to Have a Scale Model of Production in Test?** Prior to discussing the benefits of having a scale model of the production environment in test, it is important to clearly articulate what we mean when using this phrase. It is broader than simply having a fraction of the production computing power in the test environment. As an example, imagine a production environment consisting of four machines, each with four processors. Two possible test configurations may be

1. One 4-processor machine
2. Four 1-processor machines

While it is true that both of these proposed test environments have one-quarter the processing power of the production environment, both leave out complexity. The first configuration, one 4-processor machine, disregards the complexity of having the application distributed across multiple systems. The second option, four 1-processor machines, ignores the impact of having multiple processors per machine. Although the number of processors per machine does not directly affect the administrative configuration of the WebSphere environment, it does impact the ability to troubleshoot the application. More specifically, this configuration increases the difficulty of finding synchronization bugs.

Most synchronization problems are only found on multiple processor machines where true multithreading and parallelization occur. Thus, not testing on multiprocessor servers can lead to deploying an application, fraught with synchronization problems, into the production environment. A more correct scale would be to have four 2-processor machines or, at a minimum, two 2-processor machines.

Another key component to having a scale model of the production environment in test is to ensure that the entire path length of the application is tested with no parts skipped, ensuring that all tests are functionally and systematically complete. This includes having all database connections, Web servers, proxy servers, messaging servers, firewalls, and so on, configured and running in the test environment as they will in production. Failing to test the entire environment may result in deployment or run time failures.

**Benefits of a Scale Model** The most obvious reason for testing an application is for correctness. It is important to verify that the application actually works as designed, but this is only one of the reasons for testing. Another commonly overlooked reason for testing is to allow you to become familiar with the dynamics of the application, its attributes in steady state, startup, and under load in a safe environment. This is important because it will allow you to quickly identify and isolate anomalies when the application enters the production environment. If the test environment is not to scale of the production environment, then the dynamics observed will

change as the application moves to production, thus crippling your problem determination ability and putting the company in a dangerous situation.

Maximizing performance tuning efforts is yet another reason for having a scale model of the production environment in test. Performance is one of the key characteristics of any application, often the second most important next to correctness. For an application to achieve maximum performance, it must be tuned. Performance tuning is a highly iterative process dependent on many parameters set both within an application server and the wider environment. If the test environment is not to scale of the production environment, much of the hard work that went into tuning the application and its environment is wasted because the parameters that achieved the best results cannot be directly applied in production. A concrete example of a parameter that has a large effect on an application server's performance, thus an application's performance, is the Java Virtual Machine's heap size. For example, if, by tuning, it is discovered that a maximum heap size of 256 megabytes achieves the best performance in the test environment and the test environment is one-half scale of production, then this number can be doubled in production with a higher degree of confidence than if the test environment were not to scale of production.

#### 25.1.1.2 Isolated Test Environment

To ensure the accuracy of any test, it is important to mandate that the test environment be isolated from other environments and activities. This is necessary to ensure that the observed behaviors are the result of the application or services with which the application is interacting and not some external event. An isolated test environment consists of machines dedicated to running WebSphere, an isolated subnet, dedicated database, and any other external systems and resources. One of the most frustrating situations when attempting to debug a problem is being unable to isolate its source, only to later find out that the problem occurred because of some external situation.

#### 25.1.1.3 Test Scenarios

To ensure that an application is ready to be deployed in the production environment, care must be taken to ensure that the scenarios under which it is tested are realistic. Realistic test scenarios consist of two parts.

The first step in creating realistic test scenarios is to ensure that the scripts used for testing the application are realistic examples of what a user might do in production. General usage scenarios are usually known prior to developing an application in the architecture phase; however, like most things, enterprise applications are not always used the way their architects envisioned. Instead, people use the application in the way that feels most intuitive to them. Therefore, it is best to get test cases from the users themselves. There are several ways to gather this information, which vary depending on what is being deployed into production.

If the application is an update to an existing application already in production, then real-world usage can be tracked on the current production version of the application, either by logging or with an external monitoring tool capable of capturing transactions.

If the application is completely new, with functions never before available to its users, then the task of gathering real-world scenarios becomes more difficult. One option is to use IBM's

Rational Unified process or the Extreme Programming paradigm at *http://www.extreme programming.org/*, in which the end users are constantly evaluating the application as it is being developed. Another option is to verify the application's general functionality via some architected use cases then release the application to a small alpha or beta test group to gather more user oriented test cases.

**GOTCHA**

A commonly forgotten step when creating test scripts is to validate the returned results. By default, many of the stress-testing applications only check for a HTTP 200 response, which is insufficient when testing J2EE applications. If the results are not verified, the application could be displaying incomplete Web pages or, worse yet, the data for a different user.

Once the test cases have been created, the next step is to run them against the application in the test environment. As it is important to gather realistic test scenarios, it is also important to run a diverse mixture of these scenarios while testing the application. The mixture should be a representative set of your user population. For instance, if there are six different user types expected in the production environment and the majority (50 percent) are expected to be of one type (type A) while the remaining 50 percent will be equally distributed among the five remaining user types, then you will want to run the tests with a similar distribution. In this example 50 percent of the virtual users would be executing test A while the remaining 50 percent would be distributed evenly among the other test cases.

**GOTCHA**

It is important to capture and test all possible user types even if there is one user type that is only used once in a great while. An example would be an administrator performing some weekly or monthly functions. Though this administrator may be less than 1 percent of the user population, it is important to understand the implications of that user's actions.

**TIP**

While monitoring the performance of the application and its dependent parts, also monitor the statistics of the testing machine. Never let the stress testing software or hardware be the bottleneck in a test environment.

**Test Types** After the test scenarios have been created, the next step is to execute them in various test types. The first tests that should be run are those that verify the correctness of the application. There are several others tests that go beyond just testing functionality of the application which should also be run. Three of the most important tests are performance, stress, and endurance. Describing the detailed steps required to execute each of these tests is beyond the scope of this book, so we will give a brief overview of each and pointers to more in-depth resources as appropriate.

The performance test is probably the most well-known test type. In a performance test, as the name suggests, the goal is to optimize the performance tuning parameters to maximize the

overall performance of the applications being tested. The process for performance tuning WebSphere Application Server is described in great detail in chapter 22 of this book and in the whitepaper, *WebSphere 4.0 Performance Tuning Methodology* found at *http://www.ibm.com /software/webservers/appserv/doc/v40/ws_40_tuning.pdf.* The paper was written for the 4.0 version of WebSphere Application Server; however, the logic and methodology detailed is applicable to every WebSphere release, including the latest, V5.1.

One of the main themes of this chapter, thus far, has been the importance of having a scale model of your production environment in test. A key component of this is having an accurate model of user workload to ensure that the planned capacity is enough to handle the estimated user load. Stress testing is designed to apply user workload, above and beyond what is expected, in an attempt to find the breaking point of the application and its environment. This type of test has several benefits. First, it will help you better understand the characteristics of the application under high load and what piece of the application is likely to fail. If a potential weak link can be identified, procedures can be put into place to deal with the problem, allowing you to be proactive instead of reactive. The second benefit is that application bugs, which might not surface under normal load, will get flushed out. One of the most prominent types of these bugs are those caused by synchronization (or lack of it). Flushing out synchronization bugs is a function of time, load, and parallelism. The larger these components are, the greater the possibility of finding a synchronization problem. Stress testing increases the load part of this equation.

The last type of test we discuss is the endurance test. The length of the endurance test is what differentiates it from all other tests. Most tests last several minutes or, at most, an hour, but in the endurance test the application is run under heavy load, as expected in production, for many hours or even days. The purpose of this is to uncover problems that may appear after extended usage. There are many problems that fall into this category, including session problems, intermittent failures, and the aforementioned synchronization bugs.

### 25.1.2 Change Control Best Practices

The production environment should be a strictly controlled environment. Failure to adhere to a stringent change control policy can result in inexplicable problems in the production environment. This section describes some best practices associated with change control with the production environment.

#### 25.1.2.1 Restricting Administrative Privileges

The first step in ensuring proper change control is to limit the number of people who have administrative privileges on the production machines. Oftentimes we have encountered production systems on which many people in the organization have administrative privileges, leading to a myriad of problems. This is especially problematic if the production machines are shared across many departments within the organization. In most cases, each department will have their own set of priorities, and, if the administrative privileges are not restricted to one or a select few administrators, people can unwittingly make changes that impact other applications within the environment.

#### 25.1.2.2 Access History

Even with proper restriction on administrative privileges, problems can occur because of administrative errors. When they happen it is useful to have a log detailing who made the last change, at what time, and from which remote machine. This is not to place blame but rather to identify what actions caused the problem and to be able to inquire as to why that particular action was performed. Once the problematic action is identified, an alternative procedure can be created to avoid future outages.

Another reason to keep an access history is to identify hackers. By following the potential hacker's actions, you can see what techniques he/she is using to access your site, and you may be able to identify their intentions if they should break in.

#### 25.1.2.3 Backing Up Your Configuration

Prior to making any configuration change, it is a good idea to back up the current, working configuration. WebSphere provides a utility, backupConfig, to aid this process. BackupConfig is command line utility (found in the <WASND_ROOT>/bin and <WAS_ROOT>/bin directories) that creates a backup of the installation's configuration files, including application and server data. BackupConfig can be run by executing the provided executable. Be aware that its default behavior is to stop all WebSphere processes prior to performing the backup. This behavior can be overridden by supplying the switch *–nostop*. The complete command would be *backupConfig –nostop*.

**TIP**

When using WebSphere in a distributed environment, backupConfig only needs to be run on the Deployment Manager.

To restore from a previously saved configuration, use the restoreConfig command. Like backupConfig, restoreConfig's default behavior is to stop the WebSphere processes prior to restoring the configuration. Again, the *–nostop* command line option can be used to override the default behavior.

**TIP**

Unfortunately, we are all human and it is possible to forget to back up the configuration prior to making a change. Creating cron jobs, a scheduled execution of a task, is one way to overcome this human limitation. By creating a daily or weekly cron job that runs the backupConfig command, you can ensure that you have a recovery point even if you forget to manually run the backupConfig command.

Beyond backing up the configuration, it is also a good idea to backup the entire WebSphere installation directory immediately after installation, before and after any major changes, such as adding a node to a cell. This is useful in the event an administrator accidentally deletes a critical file, such as startupServer.bat or java.exe. Mistakes such as these are not uncommon, especially when working from the command line and dealing with many different directories at once.

Regardless of whether this has happened in your organization before, it is best to protect against these types of mistakes and create regular backups.

#### 25.1.2.4 Maintain a Log History

Usually a problem is discovered only after several occurrences and perhaps not until an external complaint is logged. By keeping a log history, you can uncover how long this problem has been occurring and potentially find a pattern in the events that caused the problem to occur.

#### 25.1.2.5 Documented Procedures

It is easy to forget a step when performing a complex operation such as installing WebSphere or upgrading an application. The best way to avoid such errors is to document the steps for these complex operations. The document should consist of clear and detailed steps with screen shots that show expected results to avoid possible confusions and misinterpretations. Every administrator should have a hard copy of these documents and follow them to the letter.

Execution of the procedures should involve, at the very least, two people. The first administrator executes the steps and the other verifies them.

**TIP**

Automating the documented procedures is an advanced technique that, when implemented properly, eliminates all potential user error. The risk is that there is no human control on the process. For more information on automated scripting, refer to chapter 20, Automated WebSphere Administration.

### 25.1.3 WebSphere Best Practices

IBM provides numerous white papers and articles about best practices and design patterns for WebSphere. The WebSphere Developer Domain Web site is a great place to find many useful WebSphere articles, including best practices-specific papers. A Web site that hosts an extensive collection of best practices for administering the WebSphere Application Server Web site is also available at *http://www.software.ibm.com/wsdd/zones/bp*. IBM best practices are reviewed and updated for new versions of WebSphere. It is recommended you stay informed and follow the best practices published by IBM.

#### 25.1.3.1 Application Best Practices

Application performance and scalability is heavily influenced by the design of the application, database, and other resources. The importance of following application best practices is underestimated in many cases. A good application design for performance and scalability follows certain fundamental best practices and avoids common mistakes. Following the WebSphere application best practices is a good starting point for designing applications with great performance and scalability. Performance, in particular, suffers from poorly designed and implemented applications.

#### 25.1.3.2 Code Reviews

Code reviews are an extremely effective way to find code bugs and, thus, prevent problems, but many times they are not fully completed due to project deadlines or other time constraints. Code reviews are performed by programmers who are not the authors of the code that is investigated. In many cases code reviews are as effective as or even more effective than testing.

Code reviews include simple things like checking for naming conventions, code indenting or adding Javadoc comments, and more complex tasks like code optimization or logic checking. The tasks in the first category make your code easier to maintain and reuse, while tasks in the second category check the correctness and improve the performance of your application. Code optimization like caching data, efficient use of database connections, or eliminating unnecessary object instantiation can significantly increase performance.

The good news is that you can use Java code analyzers to automate much of the code review process. Most Java code analyzers detect things like unused or duplicate imports, unused private and local variables, missing Javadoc comments, violation of naming conventions, and empty "catch" blocks or "if" statements. You will still have to check for logic errors and code optimization, but most of these simple (but time-consuming) code checking tasks are automated.

**TIP**
You can integrate Java code analyzers with Ant. In this way you can check your code any time you run a unit test or complete a build.

### 25.1.4 WebSphere Fix Packs and Interim Fixes

In many cases, the solution to your problem may be as simple as applying a fix pack or interim fix. For this reason, we recommend that you stay up to date with fix packs for your version of WebSphere and check for interim fixes when problems occur. Although applying a fix pack requires careful planning, this operation pays off in most cases by fixing or preventing problems. It is recommended that you also keep current supporting products such as DB2, and so on.

**TIP**
Fix packs should be tested in nonproduction environment first. Occasionally features in a fix pack may not be compatible with existing applications.

Interim fixes are WebSphere code fixes created for known individual problems and should be applied when you have a critical problem without a valid workaround. Interim fixes are individually tested and are integrated with the next WebSphere fix pack. Make sure that you check the WebSphere Application Server fix packs and interim fixes Web site *http://www.ibm.com/software/webservers/appserv/was/support/* when a problem occurs in order to determine if this is a documented problem. Each released version of WebSphere details the defects that were fixed in its accompanied release notes. You can also create a "My support" profile to receive weekly e-mail notifications about IBM product updates. For WebSphere, go to *http://www-306*

*.ibm.com/software/webservers/appserv/was/support/* and select My Support from the right menu.

## 25.2 Problem Determination Methodology

Often, when an error occurs in WebSphere, the user knows a problem has occurred but doesn't know what to do about it or what it means—the user just knows that something is broken. This section takes you through WebSphere problem determination methodology, a series of rules to follow when you encounter a problem, to help pinpoint its origin and take it to resolution.

Why is methodology important? The foundation of any problem determination process is good methodology. Knowing how to go about determining if you have a problem, where the problem exists, and the basics of how to solve that problem are important to any enterprise project.

You might be asking, "Why not tell us how to troubleshoot WebSphere?" Troubleshooting a product the size of WebSphere could fill a book on its own. WebSphere is a very large and complex piece of software and to try to address in detail how to debug each specific component is beyond the scope of an administration book. While this subsection does not cover detailed troubleshooting of the application server, it does lay the foundation for troubleshooting your environment if and/or when a problem occurs. This methodology section lays out a set of rules you can employ when doing in-depth problem determination. Following these rules when a problem occurs can help expedite locating the problem (the first step in any problem determination process) and then resolution.

### 25.2.1 Locating the Error in a Complex Environment

When a problem occurs, pinpointing its origin can take some detective work, especially in a complex environment where multiple tiers with multiple products are integrated with one another. Knowing where each product's log files are located is an important step in knowing your environment, since the log files often provide very useful information. We have detailed where to locate and how to read WebSphere's log files in chapter 24, WebSphere Problem Determination Tools—Logging and Tracing. Often, when troubleshooting in a complex environment, problem determination becomes a team effort, involving administrators and application developers from each component in the environment.

This is especially true in the initial stages of determining where the problem resides (e.g., which tier, which component/product). Rarely is one person an expert in each component that makes up the environment (e.g., the back-end, the Web server, the edge component, the authentication server, the application, etc.). It is often necessary to involve people from various roles to assist in determining where the problem is or is not located. For example, if you are experiencing the problem when testing a new build of an application, make sure the necessary application developers are available to help determine if the error could be originating from the application code. Or, if the error is occurring when accessing the back-end data store, involve the database administrator to assist in determining if the error is originating from the database.

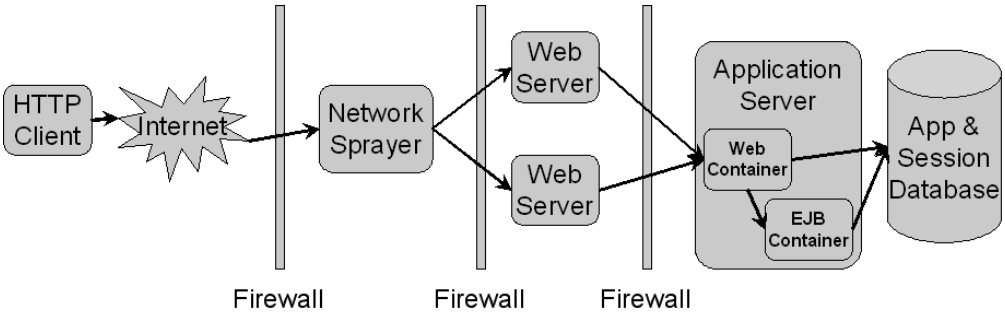


**Figure 25.1** Example request path through enterprise environment.

To help pinpoint the component where the problem exists, it is useful to create a diagram of the path a request would take, assuming it did not fail, beginning with the client all the way to the back-end, depicting each component the request touches. For example, assume that a request for a simple servlet is failing. If you were to map the flow of a request in a simple environment beginning from a browser, it might look something like the servlet request goes through the proxy firewall to the network sprayer; from the network sprayer through the domain firewall to the Web server; from the Web server through an additional firewall to WebSphere Application Server, which then responds back to the Web server that responds to the browser.

At any one of these points, the request might fail. However, the request can be tracked by looking at access and error logs of each of the components involved. Additionally, this might require enabling trace on the various components to track the requests. For example, the access log on WebSphere's embedded HTTP Server might be enabled so you can verify that the request from the Web server made it into WebSphere. Possibly, one of the components is throwing error messages into the log files, or there are communication issues between two or more of the components involved. Once the failing component(s) is located, a more thorough examination of the error can begin. Problem determination of products outside the WebSphere Application Server or products installed and running in WebSphere Application Server (like WebSphere Portal or a custom J2EE application) is out of the scope of this section. While this section is mainly geared toward problem determination with regard to the WebSphere Application Server run time, some of this methodology still applies to external products and applications.

### 25.2.2 Could the Error Be Valid?

Once the error is isolated to a particular component(s) within the environment, one of the first things to evaluate is the error code, message, and any associated stack trace that appears. Often these error codes and/or messages provide useful information as to what went wrong. Most products and protocols also have guides that provide additional information on particular error codes. WebSphere has a Message Reference guide that is a subsection of the InfoCenter documentation. This Message Reference section has a description of each error code that WebSphere can log in

**Table 25.1** IBM Product and Protocol Message Reference Guides

| Product or Protocol | Message Reference Guide | Where to Locate |
|---|---|---|
| WebSphere Application Server | Message Reference Section of Info Center Documentation | Online InfoCenter Documentation: *www.ibm.com/software/webservers/appserv/infocenter.html* |
| DB2 Universal Database | Message Reference, vol. 1 and 2 Guides | Online DB2 Core Documentation: *www.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/v8pubs.d2w/en_main* |
| IBM Http Server | Troubleshooting Section of InfoCenter Documentation | Online Infocenter Documentation: *www.ibm.com/software/webservers/httpservers/doc/v1326/manual/ibm/index.html* |
| Apache Server | Online Documentation and FAQ | Online Apache Documentation Project: *httpd.apache.org/docs-project/* |
| Hypertext Transfer Protocol (HTTP) | Status Code Section of RFC2616 | Online RFC 2616: *www.w3.org/Protocols/rfc2616/rfc2616-sec10.html* |

its trace or log files. For information on how to read or locate WebSphere error codes and messages, please refer to chapter 24. Also, in Table 25.1, we are providing information on where to locate additional commonly used IBM product documentation, as well as protocol error codes.

Sometimes, pertinent information can be located in different product log files, which can be aligned by timestamp. For example, an exception occurring in the database could provoke error messages to be logged in the application server log files, as well as the Web server logs. Once you locate one error message, you can use its associated timestamp to cross-reference the database, application server, Web server, log files, and so on. This is also a good technique to use when locating the root of the error.

**TIP**

It is important to have the system clocks synchronized on each machine such that time stamping cross-referencing can be easily used. If the system clocks are not synchronized, the logs can still be cross-referenced; however, the times must be skewed appropriately.

When tackling a problem, it is always good to first assume that the error is valid before declaring that there is a bug or a defect in the running product. The error code and associated message can often be enough for you to diagnose and fix the problem.

For example, let us take a look at a fix pack installation problem scenario. Upon installing a fix pack onto an existing WebSphere configuration, an error occurs preventing the installation. The initial reaction to such a failure could be to assume that there is a defect with the installation or the WebSphere run time. Listing 25.1 depicts a portion of a reproduced log file with the problem.

**Listing 25.1** Portion of log file from failing installation of a WebSphere fix pack.

```
...
Results:
================================================================
Time Stamp (End)    : 2003-07-15T17:08:42-04:00
EFix Component Result : failed
EFix Component Result Message:
================================================================
WUPD0239E: Fix removal failure: The processing of fix
WAS_WSADIE_ND_01_16-2003_5.0_cumulative, component prereq.wsadie
failed. See the log file
C:\\WebSphere\DMgr\properties\version\log\20030715_210842_WAS_WSADIE_ND
_01-16-2003_5.0_cumulative_prereq.wsadie_uninstall.log for processing
details.
================================================================
EFix Component Installation ... Done

Exception: WUPD0223E: Fix uninstall failure: The update for component
{1} for fix pre-req.wsadie could not be installed .

...
```

As you can see from the log file excerpt, an exception occurred that prevented the installation. This log file also referenced an additional file for more information (see the highlighted portion of the log file above). Upon investigation of the referenced log file, 20030715_210842_WAS_WSADIE_ND_01-16-2003_5.0_cumulative_prereq.wsadie_uninstall.log, additional information about the problem is uncovered. Listing 25.2 shows a reproduced portion of the referenced log file, 20030715_210842_WAS_WSADIE_ND_01-162003_5.0_cumulative_prereq.wsadie_uninstall.log.

**Listing 25.2** Portion of log file from failing installation of a WebSphere fix pack.

```
...

2003-07-15T17:08:42-04:00 Applying entry 1 of 5  20% complete
2003-07-15T17:08:42-04:00 Preprocessing entry (restore):
```

```
2003-07-15T17:08:42-04:00 No EAR processing noted.
2003-07-15T17:08:42-04:00 Next entry name: lib/jdom.jar
2003-07-15T17:08:42-04:00  entry path: C:\WebSphere\DMgr\lib\jdom
.jar
2003-07-15T17:08:42-04:00 Error 16--File could not be deleted:
C:\WebSphere\DMgr\lib\jdom.jar
2003-07-15T17:08:42-04:00 Fetching entry ...
2003-07-15T17:08:42-04:00 Preprocessing entry (restore):
2003-07-15T17:08:42-04:00 No EAR processing noted.
2003-07-15T17:08:42-04:00 Next entry name: lib/marshall.jar
2003-07-15T17:08:42-04:00 entry path: C:\WebSphere\DMgr\lib\
marshall.jar
2003-07-15T17:08:42-04:00 Error 16--File could not be deleted:
C:\WebSphere\DMgr\lib\marshall.jar
2003-07-15T17:08:42-04:00 Fetching entry ...
2003-07-15T17:08:42-04:00 Preprocessing entry (restore):
...
```

Again, we have highlighted the errors in the log file excerpt—you can see that some of the jar files being replaced during the installation of the fix pack could not be removed.

**TIP**
A log file can have a tremendous amount of information in it. Sometimes searching for "rror" or "xception" can help pinpoint problems easily. Notice in both search string the "E" was left off such that capitalization does not limit the search.

Since these jar files could not be removed, the installation was failing. With this information in hand, we can begin to diagnose the problem—first assuming that the error was valid. Why couldn't the files be removed? The following options could all be valid possibilities:

- The jar files did not exist in the first place.
- The person running the installation did not have the appropriate permissions to remove files on the operating system.
- The files were locked by a running process.

After validating that the jars did exist and the installer had the appropriate permissions, the last option was investigated. A quick check of all running processes uncovered that WebSphere was still running while the fix pack was attempting to be installed. Since WebSphere was still running, it had locked the jar files to prevent run time corruption. So, in fact, the problem was not a defect or bug in WebSphere's installation of the fix pack; instead it was a valid response to an invalid operation (note that the fix pack installation directions require all WebSphere processes to be stopped before running the installation program). Once all WebSphere processes were stopped, the fix pack installation was successful.

### 25.2.3 What Has Changed?

When an error occurs, another technique to help pinpoint the root of the problem is to determine what might have changed to invoke the error. For example, did the error occur just after you ran a new test scenario, or did the error begin after you adjusted some TCP configurations on your operating system? Rarely does an error "just begin happening" if nothing was changed in the environment (network, operating system, application, server configuration, etc.). Therefore, it is an important part of problem determination to uncover what might have changed to invoke the problem at hand.

In an earlier section, Change Control Best Practices, a change control process was detailed as a best practice for problem prevention. If this system is in place and adhered to, determining if something in the environment has changed becomes much easier. Also, note that when we say "environment" we do not just mean WebSphere administration. The environment encompasses much more than this—it includes items such as the operating system settings, application configuration and code, test cases, configuration of supporting products either running on WebSphere or communicating with WebSphere, such as the Web server, back-end, authentication server, portal server, edge components, and so on. Determining if something has changed can be more than asking yourself if you have recently altered a configuration setting (unless you are the only one with administrator privileges to every server in the environment). Communication, therefore, is the key, especially in a complex environment. We are often surprised how, in some environments, communication between the administrators of various components (WebSphere, Database, network, etc.), as well as with application development, is minimal. Often, an administrator will call a product support line before calling a coworker in a different department to see if they might have altered a configuration.

Pinpointing the change does not mean that a bug does not exist, nor does it mean that the problem is now solved. Often there is a good reason for the change that has been effected. However, knowing what the change is and how it affects the system is important in finding a solution (whether it is a product bug fix, a configuration tweak, etc.).

### 25.2.4 Simplify, Simplify, Simplify

When you are running in a complex environment and an error occurs, finding the problem can sometimes be equated to finding a "needle in a haystack." With so many different products and configurations involved, solving the problem becomes like solving a multiple variable algebra problem: the greater the number of variables involved, the more complex it is to solve.

Also, there is not always just *one* item that causes a problem to occur. Rather, it can be a combination of settings, coupled with a particular path through running application code, that triggers the error. To help limit some of the variables in the problem, it is wise to strip the environment back to the simplest possible running environment in which the error still occurs.

The best problem determination environment is one where the error can be reproduced with the *simplest* test scenario, running the *simplest* application code, deployed in the *simplest* environment. In this environment, not only is it easier to describe the problem to support (if

necessary), but also it limits the number of variables involved in the problem, making it easier to determine a solution.

#### 25.2.4.1 The Simplest Test Scenario

Evaluate the test scenario that prompts the error to occur. If the test scenario is testing multiple conditions, can the test be limited to only the condition that fails? By narrowing what is tested until you have located the simplest test scenario that still causes the failure, you can save time when rerunning the test scenario, as well as narrow the number of variables when reproducing the test. You might discover that it is the sequence in which the tests are run that causes the problem, and/or eliminate the components that appear to not related to the failure.

Additionally, if the failure is occurring during load testing, try to find the minimum amount of load that still reproduces the problem. For example, if the test does not fail with a single user, but fails with two users, there might be a thread synchronization error. If the tests only fail under high load, it might be that your application or environment needs to be tuned for performance (please see chapters 21 through 23 of this book to learn about performance as relates to WebSphere).

#### 25.2.4.2 The Simplest Application

When running a complex application, it is often difficult to determine whether the source of the error resides in the running application, in the WebSphere run time, or in some other area. If you can eliminate the running application in the simplified environment by reproducing the error with an alternate, much simpler application, that is very beneficial. You can eliminate the enterprise application as the source of the problem by attempting to reproduce the error with one of the IBM WebSphere sample applications that are installed with WebSphere or by creating a very simple application that forces the error to occur.

**TIP**

The technique of using an IBM WebSphere sample application or a simple sample application that can reproduce the problem is especially beneficial when working with IBM support. If using a created simple application, include it with any documentation that is provided to WebSphere support, with a description of what it does and the error it causes. This can help expedite support's interaction in determining the problem.

#### 25.2.4.3 The Simplest Environment

To locate the origin of the problem, either a product or a WebSphere component level, it is best to reproduce the problem with the simplest configuration possible. For example, if the problem is occurring with a Web application, remove the Web server from the environment by accessing the application directly via WebSphere's embedded HTTP server. If the problem is with persistent Enterprise Java Beans (EJBs), try to manually invoke some of the update or select queries on the back-end to validate that they run correctly. Some other suggestions for simplifying the problem determination environment include

- Disabling work load management (distributed only)
- Disabling security
- Disabling the JIT compiler
- Moving the test clients onto a machine that has a direct route to WebSphere (rather than having to go through firewalls or edge components)

### 25.2.5 Do You Have Enough System Resources?

When failures begin to occur during load tests, sometimes it is not due to a run time failure, but rather a potential performance issue. Please refer to the Part 5, WebSphere Performance, of this book for additional information on performance monitoring and tuning. However, do remember that every machine will have its limits. In some situations, additional hardware will be needed to support particular load requests.

Performance monitoring can also uncover problems such as memory leaks that can severely impact application performance. It is highly recommended to tune your application before releasing it in a production environment.

### 25.2.6 What to Do If the Problem Is in Production

When a failure occurs in a production environment, it is often a critical situation. If you believe the problem to be a WebSphere run time defect, it will be important to contact IBM WebSphere support (1-800-IBM SERV) immediately so they can begin investigating the problem. It is also important to make sure that no information surrounding the failure is lost. It is a best practice to backup all log files, including database and Web server logs, if applicable, so they can be referred to later, if necessary. Until a solution is found, rollback or disable any change or update that might have invoked the problem. In parallel, it is pertinent to try to reproduce the problem in a test environment. A problem that can be reproduced in test will lend itself to easier problem determination since detailed traces and logging can be enabled without fear of affecting performance or up-time in the production environment. It also provides an experimental environment for being able to freely alter configuration parameters, as well as providing a simpler, less complex problem determination environment.

**TIP**

When the cause of the problem is determined, use it as a lesson learned. It is important that the failing scenario works its way back into the test suite that is run before any application is released in production. This way, the problem can be prevented in the future. Be sure to update procedures and test cases to avoid this problem in the future.

### 25.2.7 Where to Go for Help

IBM has an extensive WebSphere support Web site that contains self-help and problem submission information. This page should always be used before contacting IBM support. The WebSphere support Web site is accessible at *http://www.ibm.com/software/webservers/appserv/was/support/*.

The self-help section of the WebSphere support Web site contains links to several online resources meant to help you troubleshoot a WebSphere problem. Using this site, you can search on keywords to find Frequently Asked Questions (FAQs), Technotes, Hints and Tips, and other documents that address existing WebSphere problems. FAQs document common problems and solutions. Hints and Tips contain information about installing, configuring, and troubleshooting WebSphere. Technotes are documents containing customer-reported problems and solutions. You can also download WebSphere tools and utilities, as well as WebSphere fix packs and interim fixes. The support page also contains links to educational material such as IBM online publications, redbooks, and white papers.

The WebSphere InfoCenter is another resource for self-help. The InfoCenter is available online at *http://www.ibm.com/software/webservers/appserv/infocenter.html* or it can be downloaded as a PDF file. The local version of the InfoCenter is also available as an Eclipse documentation plug-in and can be downloaded from *http://www.ibm.com/software/webservers/appserv/infocenter.html.* To view the local documentation, you also need to install the IBM WebSphere Help System, which is a viewer for displaying product or application information developed as Eclipse documentation plug-ins. The IBM WebSphere Help System is built on open source software developed by the Eclipse Project. The InfoCenter contains a problem determination section, and you can also search the InfoCenter using keywords.The developerWorks Web site contains very good information for WebSphere developers in the section dedicated to WebSphere, which is available at *http://www.ibm.com/developerworks/websphere/*. The WebSphere developerWorks is a great source for articles and best practices related to WebSphere products. The site also provides other features like code downloads, technology previews, and forums.

Other helpful WebSphere resources are the WebSphere newsgroups and WebSphere usergroup forums. There are several such newsgroups and forums, and they usually contain very useful information provided by WebSphere users. Some of these newsgroups are monitored by IBM personnel, helping to ensure the integrity of information included within those newsgroups.

WebSphere Studio Application Developer and Site Developer V5.1 have a new feature that allows you to search on keywords for several products, including WebSphere Application Server. This new feature is provided in the form of several product-specific plug-ins. The search is performed on resources like the WebSphere support Web site, the WebSphere InfoCenter, and Google newsgroups. Access to these resources is provided from one central product-specific page. Besides search capabilities, the plug-ins also provide a collection of local documents for self-support. These documents are copies of FAQs, Technotes, Hints and Tips, and other resources that are frequently used by WebSphere support personnel. One advantage of having these local documents is that they are searched when you perform a search through the WebSphere Studio Application Developer or Site Developer Help menu. To access the product plug-ins, select Help → Help Contents from the main menu, and then click on Support information of the left side of the page. Please see Figure 25.2.

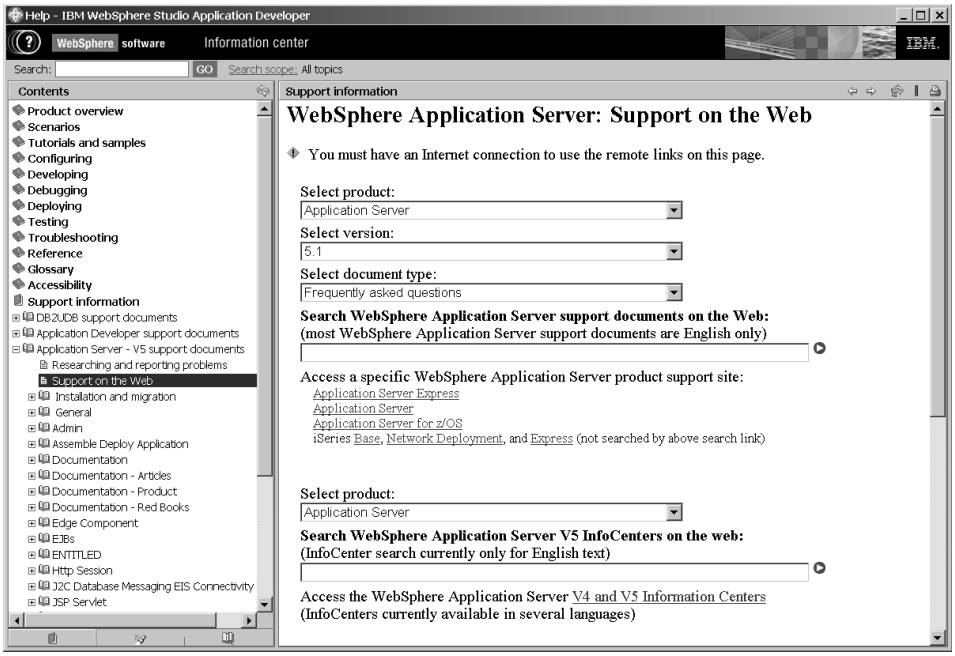


**Figure 25.2** WebSphere Studio Application Developer Support information.

## 25.3 Working with IBM WebSphere Support

There are times when you need to work with IBM WebSphere support personnel to troubleshoot a WebSphere problem. The goal of this section is to familiarize you with the process of resolving a WebSphere problem with the help of IBM WebSphere support. We will tell you how to open a Problem Management Record (PMR), what information to have ready, and when you should involve IBM support to resolve a WebSphere problem.

### 25.3.1 When to Involve WebSphere Support

You have already tried to solve the problem by yourself. By now, you have searched the WebSphere support Web site, newsgroups, WebSphere Developer Domain, and other resources in order to determine if this is a known problem. You have also checked the latest WebSphere fix packs and interim fixes. Unfortunately you haven't found a match. It's time to engage IBM WebSphere customer support.

### 25.3.2 How to Open a PMR

There are two ways to open a Problem Management Record (PMR). You can submit an electronic PMR using the Electronic Service Request (ESR) tool, or you can call IBM customer support (1-800-IBM-SERV). ESR is a Web-based problem submission tool and is available from the

problem submission page at *http://www.ibm.com/software/support/probsub.html*. In order to use the ESR tool, you have to be enrolled in the IBM Passport Advantage Program and be registered as an authorized caller. Authorized callers are registered by the site technical contact, a person in your company who is responsible for maintaining the list of persons (in your company) authorized to use the ESR tool for problem submission. Once your site technical contact adds you as an authorized caller, you will be able to use the problem submission page to create a user ID and password. The user ID and password will be required to access the ESR tool. The ESR tool allows you to open new PMRs, as well as work with your existing PMRs.

**TIP**

The IBM Customer Number or Customer ID consists of 7 or 10 digits and is used to identify an IBM Passport Advantage support contract for a customer. Your company may have several contracts. Check with your site technical contact to determine the correct IBM Customer Number or Customer ID to use.

The following steps are required to submit a new PMR:

1. Click on your customer number.
2. Click on the Report a New Problem button.
3. Select the product for which you want the PMR to be created.
4. Select a component from the drop down list.
5. If necessary, edit the contact information in the Report a New Problem page.
6. Complete the rest of the fields in the Report a New Problem: Environment, Severity, Problem description, etc.
7. Click on Submit Problem Report. A page confirming that your PMR has been submitted to a support queue will be displayed.

Once the queued PMR is processed, a PMR number will be assigned to it and e-mailed to you. Keep this PMR number handy since you will need it every time you update the PMR or talk to IBM support.

**TIP**

The list of products and the list of components for which you can open PMRs can be quite long. You can use the Product Search and Component Search functions to find a specific product or component.

**TIP**

You can create and update a user profile in the ESR tool, with personal information like your phone number, e-mail, pager, and so forth. This profile will be used when you submit a PMR via the ESR tool. To access your profile, go to Update Maintenance Agreements from the main support page.

Figure 25.3 illustrates the ESR page that allows you to create a new PMR.

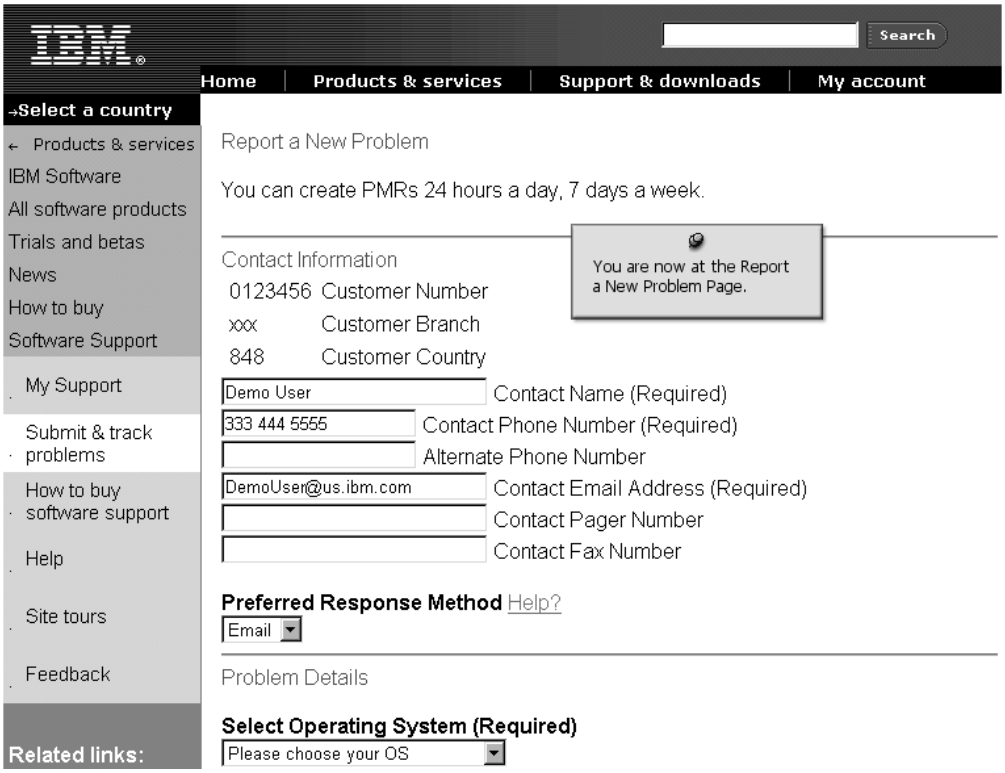


**Figure 25.3** ESR page for creating a new PMR.

The other way to submit a PMR is to call IBM customer support (1-800-IBM-SERV). When you call IBM customer support, you will also need information such as your company's customer number, product, component, and so forth, similar to the information required by the ESR tool.

### 25.3.3 What Information to Have Ready

No matter how you open the PMR, electronically or by phone, you will need to provide information that will help the support personnel resolve your problem. Needless to say, you should be as specific as possible when describing the details of the problem. If you can re-create the problem, make sure you document the steps for IBM support. Also, gather as much background information as possible. For example, don't forget to specify what your environment is, what WebSphere fix packs you have installed, what Operating System and version you run, what changes you made to the system, and so on. If any messages or errors were logged, send these logs to IBM support. Any relevant information will help the IBM support personnel resolve your problem as quickly as possible. Basic background information is often missing from the PMR description, and this can significantly increase problem resolution time.

Another important piece of information you will be required to provide is the severity of the problem. Use the business impact as measure for determining the problem severity. For example, the most severe problems, those having a critical business impact, should be assigned a severity 1, while minor problems should have a severity 4.

It may also be useful to run the collector tool and have the output jar file ready to send to the IBM support personnel. For more information about the collector tool, refer to chapter 7, Getting Started with WebSphere—An Overview.

### 25.3.4 What to Expect

If a WebSphere defect is found, then an Authorized Program Analysis Report (APAR) is created for this problem. If a workaround can be implemented while the APAR is resolved, and before the fix is delivered, then IBM Support will provide instructions on how to do this. Note that making a fix available requires time-consuming operations such as comprehensive testing, packaging, and so on.

## 25.4 Summary

You should now be familiar with the following concepts:

- How to create an appropriate test. Specifically, it should be a scale model of the production environment.
- Which test types should be run to minimize the risk of encountering a problem in the production environment. These test types are correctness tests, performance tests, stress tests, and endurance tests.
- How to generate realistic test scenarios and proper workload mixes. These should be derived from real-world users.
- How to enact proper change control by restricting administrative privileges, logging administrative accesses, regularly creating backups of your configuration, keeping a log history, and documenting procedures.
- When locating an error in a complex environment, it is useful to diagram the path of the request, beginning with the client all the way to the back-end.
- Once an error has been located, first assume that the error is valid until it can be verified that it happened under erroneous circumstances.
- When an error occurs, try to determine what, if anything, might have changed in the environment that could have contributed to the condition which promoted the error.
- Try to re-create the error with the simplest test scenario, the simplest application, and the simplest environment.
- WebSphere interim fixes and fix packs.
- What resources are available for WebSphere support.
- Working with IBM WebSphere support to resolve a WebSphere problem.

# Index

## D

## J

## N

## U